1990-12

Interface-driven software development tool

Kim, Heung-Taek

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTERFACE-DRIVEN SOFTWARE
DEVELOPMENT TOOL

by

Heung-Taek Kim

December, 1990

Thesis Advisor: C. Thomas Wu

Approved for public release; distribution is unlimited.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release: distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) CS
7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
INTERFACE-DRIVEN SOFTWARE DEVELOPMENT TOOL

12 PERSONAL AUTHOR(S) Kim, Heung-Taek

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	December 1990	103

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Visual Interface, Database Management System, Human-computer interaction, Prototype

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

We live in an age where the volume of paper-based information is steadily expanding. Personal computers have a great potential as tools for managing information. Effectiveness of using personal computers is determined by how easy it is to use them, since majority of the end-users are not computer experts. Compared with the advances in software design, the important issue of computer interface has begun to be addressed recently. There has been a research joining the database with the graphical interface to give users an easy-to-use method for accessing the database. With this, users navigate through the database by following the links from one piece of information to the next. There are several classes of softwares (languages) to build visual user interfaces: traditional, object-oriented, and interface-driven languages. In this thesis, we used an interface-driven software named Guide to build a prototype visual user interface to analyze the effectiveness of interface-driven software.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL C. Thomas Wu	22b TELEPHONE (Include Area Code) (408)646-3391	22c OFFICE SYMBOL CS/Wq

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFED

Approved for public release; distribution is unlimited.

INTERFACE-DRIVEN SOFTWARE DEVELOPMENT TOOL

by

Heung-Taek Kim
Captain, Republic of Korean Army
B.S., Korea Military Academy, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1990

Author: [redacted]
Heung-Taek Kim

Approved by: [redacted]
C. Thomas Wu, Thesis Advisor

[redacted]
Myung W. Suh, Second Reader

[redacted]
Robert B. McGhee, Chairman, Department of Computer Technology

ABSTRACT

We live in an age where the volume of paper-based information is steadily expanding. Personal computers have a great potential as tools for managing information. Effectiveness of using personal computers is determined by how easy it is to use them, since majority of the end-users are not computer experts. Compared with the advances in software design, the important issue of computer interface has begun to be addressed recently. There has been a research joining the database with the graphical interface to give users an easy-to-use method for accessing the database. With this, users navigate through the database by following the links from one piece of information to the next. There are several classes of softwares (languages) to build visual user interfaces: traditional, object-oriented, and interface-driven languages. In this thesis, we used an interface-driven software named Guide to build a prototype visual user interface to analyze the effectiveness of interface-driven software.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express sincere thanks to those who helped me in every aspect. I would like to present great thanks to my thesis advisor Professor C. Thomas Wu, and the second reader Professor Myung W. Suh for their help to finish this thesis.

I would like to thank my family, especially my beloved wife In-Sook Lee for her ungrudging assistance, my big boy Hong-Bae Kim, and a cute little girl Da-Hae Kim.

And I would like to thank all my families and acquaintances staying in Korea praying for me and my family.

Finally, I would like to appreciate my country, Republic of Korea, and the United States of America which have given me a chance to study at Naval Postgraduate School.

I. INTRODUCTION

A. BACKGROUND

We live in an age where the volume of paper-based information is steadily expanding, yet our capacity for dealing with it has not kept pace. Personal computers have a great potential as tools for managing these information. Effectiveness of using personal computers is determined by how easy it is to use them, since majority of end-users are not computer experts. The potentials of this amazing machine are not limited by its power to compute, but rather by its power to communicate with its human users. Compared with the advances in software design, the important issue of computer interface development has begun to be addressed only recently. There has been a research joining the database management systems with the graphical interfaces to give users a more convenient, comfortable, and easy-to-use method for accessing database systems. The database management systems developed so far have interfaces dealing only with text information. This new approach can deal with not only text information but also graphics, sound and other types of media that can be attached to computer systems.

B. SCOPE OF THESIS

This thesis explores a framework for the visual interface for accessing databases and develops a generic visual interface which can be used to navigate through information systems. In the course of developing the implementation, we tailored this interface to a

real database system, Superbase 4 developed by Precision, Inc., and could communicate with that database through this interface. But, in this thesis, we are only concerned with the development of generic visual interface: we do not discuss the specific implementation issues related to any specific database system.

C. THESIS ORGANIZATION

Chapter II consists of five sections. In the first section, we will discuss about the visual interface. In the second section, we will introduce related works. And in the third section, we will introduce the concepts of the object-oriented language. And in the fourth section, it will talk about the benefits of building interfaces based on object. In the fifth section, we will discuss our approach.

Chapter III consists of three sections. In the first section, we discuss about a Guide's working environment. In the second section, some strong points using the hypermedia software, Guide, will be introduced. And in the third, we compare three different classes of languages used for building user interfaces.

Chapter IV consists of three sections. In the first section, we will discuss the rules or guidelines in developing user interfaces based on human factors engineering. In the second section, we will explain the principles that has been developed as guidelines to build visual user interfaces. In the third section, it introduces more principles that the author has developed during the implementation of the thesis.

Chapter V consists of four sections. In the first section, it talks about the interface-driven software development tools. In the second section, it talks how we

ported some of works from other system. In the third section, the details of development methodology will be discussed. And in the fourth section, a sample session will be provided.

Chapter VI concludes with a summary discussion of the thesis and possible future researches.

II. VISUAL INTERFACE FOR DATABASE

A. BACKGROUND

The easy-to-use, direct manipulation interfaces let the user operate directly on the objects that are visible on the screen, performing rapid, reversible, and incremental actions. Unlike traditional information retrieval systems in which users access information using Boolean operations on keyword strings, users of graphical user interface are free to move between arbitrary chunks of information by simple click of a mouse button. Users navigate through the information database by following the links from one piece of information to the next. Such an architecture encourages users to find information by following a meaningful path from one chunk of information to another until they reach their objective.

The research on a visual interface for database has been motivated by the lack of an easy-to-learn and easy-to-use query facility for accessing databases, although relational query languages such as SQL and QUEL are much better languages than those for network and hierarchical systems. [Ref. 16]

For database systems to become fully useful as information managers, end-user participation is indispensable. End users of databases typically have a good understanding of their application environment, but little familiarity with database technology. Therefore, we must rely on data processing professionals to develop an application

software. Such arrangement is prohibitively expensive for the coming decades of information explosion.

Thus we need a different way of user-interaction tool for data definition and manipulation. The key to the coherent interface must be a simple visual representation of database [Ref. 16]. Hence the users can visually interact with the system with ease.

B. RELATED WORKS

1. GLAD

One of the current visual-interface system is GLAD (Graphics LAnguage for Database) which was developed by Professor C. Thomas Wu at Naval Postgraduate School in Monterey, CA. GLAD has facilities for data definition, minimal data manipulation, on-line help system, and is able to store and manipulate graphic images as a part of database [Ref. 17]. Figure 2.1 shows one of its screen dump.

GLAD has been developed and implemented using object-oriented programming environment, and Actor as its language.

2. ARGOS

This implementation has been developed based upon the concept of paperless information management being developed for the United States Navy guided missile frigate. It presents graphical representations of equipment such as pictures and other interface factor such as audio to supplement the text information contained in the database. It was developed on Apple Macintosh using Hypercard as its environment and Hypertalk as its language. Hypercard provides a set of tools to support rapid prototyping

and implementation ideas.[Ref. 6] Figure 2.2 shows a sample session of the Argos implementation.

C. CONCEPT OF OBJECT-ORIENTED LANGUAGE

In object-oriented language, all conceptual entities are modeled as objects. An ordinary integer or string is as much as an object as is a complex assembly of parts, such as an aircraft or a submarine. An object consists of some private memory that holds its state. The private memory is made up of the values for a collection of instance variables. The value of an instance variable is itself an object and therefore has its own private memory for its state.

The behavior of an object is encapsulated in methods. Methods consist of code that manipulates or returns the state of an object. Methods are a part of the definition of the object. Methods, as well as instance variables are not visible from outside of the object. Objects can communicate with one another through messages. Messages, together with any arguments that may be passed with messages, constitute the public interface of an object. For each message to be understood by an object, there is a corresponding method that executes the message. An object reacts to a message by executing the corresponding method and returning an object in response.

A system such as databases may contain an even larger collection of objects. If every object is to carry its own instance variable names and its own methods, the amount of information to be specified and stored can become unmanageably large. For this reason, similar objects are grouped together into a class. All objects belonging to the same class

are described by the same instance variables and the same methods. They all respond to the same messages. Objects that belong to a class are called instances of that class. A class describes the form (instance variables) of its instances and the operations (methods) applicable to its instances. Thus, when a message is sent to an instance, the method that implements that message is found in the definition of the class. The instance variables and methods specified for a class are shared (inherited) by all its subclasses (children in parent-child pair of a class hierarchy).

D. BUILDING INTERFACES BASED ON OBJECTS

An object-oriented architecture has been shown to be suited to interface construction [Ref. 8]. Objects are natural for representing the user interface element and supporting their direct manipulation. The experience shows that, compared with a procedural implementation, user interfaces implemented in object-oriented approach are significantly easier to develop and maintain [Ref. 14]. The user communicates with the system, which is represented on the screen as a world composed of active objects. Each screen object has its visual representation which defines its appearance, its relation to other screen objects, and a functional role which governs its behavior.

E. OUR APPROACH

To develop a visual user interface in this thesis, we depend on Guide as a working environment and Logiix as a programming language, developed by OWL International Inc.

Guide is designed around an object-oriented information model which presents

information as a series of linked "objects" and manages the relationships between them. Every component in a Guide document, whether it is a single word, phrase, paragraph or graphic, is represented as an object. Guide and Logiix are not a true object-oriented environment and language, since it does not possess a clear and explicit concept of class, inheritance, method, and message which were detailed in the above section. Guide provides a variety of object types which perform different functions such as buttons. Using a mouse, users click on buttons to display linked objects. Using the familiar document metaphor, information in Guide is structured so that users can select the subjects to display and the level of detail appropriate for their particular needs.

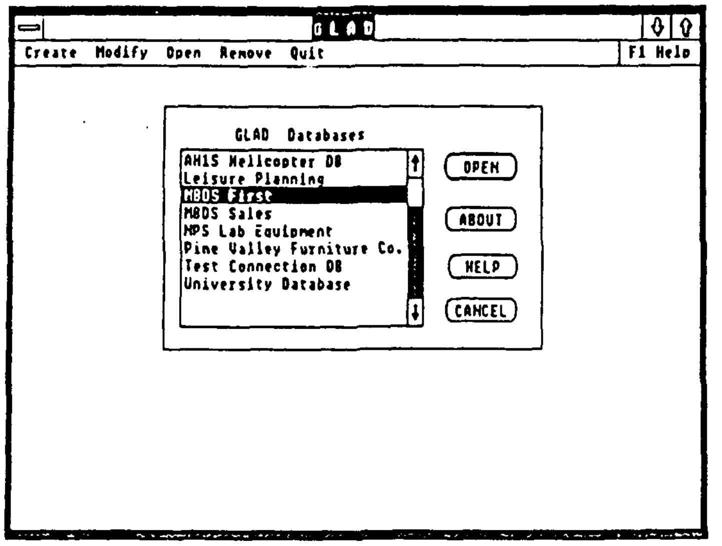

Figure 2.1 GLAD

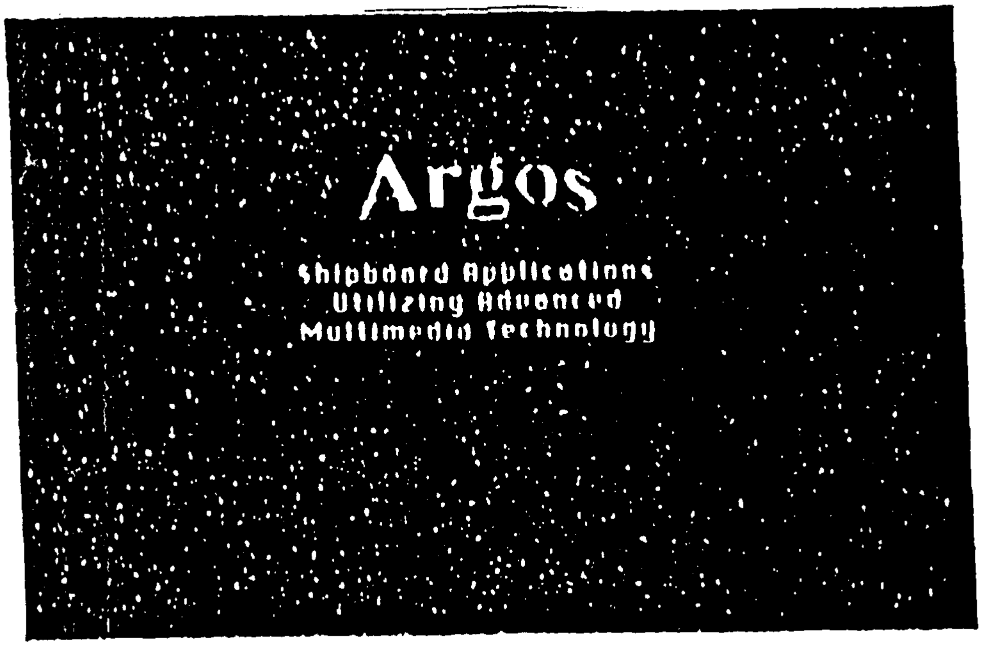

Figure 2.2 ARGOS

III. GUIDE AND ITS STRENGTH

A. GUIDE AND ITS WORKING ENVIRONMENT

1. MS-Windows

Windows is a visual extension of MS-DOS that layers itself upon DOS to provide users with a friendly, graphical interface. Figure 3.1 shows a graphic depiction of this concept [Ref. 17]. In Windows, users can execute multiple programs simultaneously in an integrated environment which gives a consistent interface to all the applications. Windows does not require users to memorize command line commands and their syntax. This helps users learn all MS-Windows applications at dramatical speed.

2. System Requirements

To run Guide version 3.0, the system must meet the following requirements [Ref. 8]:

* A high-resolution enhanced graphics adapter(EGA) or VGA display(color display is recommended).

* MS-DOS version 3.1 or higher.

* MS-Windows version 2.03 or later.

* A hard disk with at least 2MB of free space.

* At least 640 KB of memory.

* The printing device specified when installing MS-Windows.

* A pointing device such as mouse which is compatible with MS-Windows.

B. FEATURES OF GUIDE

Conventional textual documents are limited by a left-to-right, top-to-bottom sequential structure. By contrast, Guide documents allow users to move from one topic(object) to another in a non-sequential fashion. [Ref. 8]

Guide documents contain objects. Objects can be single words, sentences, paragraphs, a graphic, or even a collection of graphics. Just about anything that may be selected with the mouse can be made into an object. Objects in Guide are composed of three components: data,presentation attributes, and behavioral attributes. The data component is the text or graphic that appears on the screen when the object is displayed. Presentation attributes apply primarily to text objects determining how the data is displayed such as text styles, color, and so on. Behavioral attributes define the events that take place when an object is displayed or activated using the mouse.

Guide objects can be linked together. The link starts with one object, the source, in one file and finishes at another object, the target, in another file. Links are automatically created between *expansion buttons* and *expansions* when objects are defined. For *note, reference,* and *command buttons*, link must be created manually.

Certain objects that are live and can be activated with the mouse are called buttons. Guide documents can have four different types of buttons: reference buttons, expansion buttons, note buttons, and command buttons. Reference buttons present cross-reference

information in the same document or in another document. Note buttons present information in a temporary pop-up window, which is useful for providing supplementary information such as footnotes or definitions of terms. Command buttons pass instructions contained in a *definition* to executable files called interpreters, LAUNCH3.EXE, OPCL3.EXE and SERIAL3.EXE. Users can launch any applications (either MS-Windows or non-MS-Windows) with the launch interpreter, LAUNCH3.EXE. Users can control any serial peripherals connected to the serial port of the computer with the interpreter, SERIAL3.EXE. OPCL3.EXE is an interpreter which allows users to use certain commands, such as open or close, in the definition of a command button. Expansion buttons present more detailed information being "hidden" behind them. The information hidden behind them is called the expansion which also contains either text or graphics, or both.

Guide provides a transparent graphic element, called an *invisible*, with which users place a button on a background graphic in a specific location.

Users can include graphics in a Guide document. The following types of graphics can be imported into Guide: bit-mapped files (BMP), MS-Windows metafiles (MTF), tagged image file format (TIFF) files, and PC-Paintbrush files (PCX, PCC). There are two kinds of graphics: external and internal graphics. An external graphic has a link from a document. An internal graphic is a part of document being copied into it without being linked. Guide keeps the information about a graphic element, such information as whether it is external or internal, and so on.

Documents have a natural hierarchical structure that divides topics into related groups, such as chapters, sections, and so on. *Frames* provide a way to structure a document by dividing it into an easily manageable chunk of related information.

Users can easily change the way how windows in Guide looks on the screen with options. They can also arrange the windows neatly on the screen in a cascading formation.

The *group* is a way of making objects mutually exclusive. Grouping is an effective way to minimize the amount of data readers must shift through at one time.

Users can use the Guide *glossary* to store items that they use frequently such as objects such as buttons, and, nearly anything they use frequently reducing the amount of time they spend on repetitive activities [Figure 3.2].

A *control panel* is a Guide document containing buttons (icons) that control the display of information giving users options for the actions such as moving forward or backward through frames, and so on [Figure 3.3].

C. LANGUAGE COMPARISONS IN TERMS OF VISUAL INTERFACE

This section will discuss the advantages and disadvantages of different kinds of languages in terms of their use to build visual interfaces.

1. Traditional Languages

Traditional languages, such as Fortran and Pascal, state what should happen rather than how to make it happen. The interfaces supported by traditional languages are usually form-based. The user types text into fields or selects options with menus or

buttons. There are also often graphical output areas for use by applications. The application is connected to the interface through global variables that are set and accessed by both the application and interface. The advantage of traditional language-based interfaces is that they free the designers from worrying about the sequence of events, so they can concentrate on the information that is passed back and forth. The disadvantage is that they support only form-based interfaces. Hence other interfaces must be hand-coded in the graphical area provideu to applications. They also provide only preprogrammed, fixed types of interactions. [Ref. 14]

2. Object-Oriented Languages

Object-oriented languages, such as Smalltalk and Actor, provide an object-oriented framework in which the designer programs the interface. Typically, there are high-level classes that handle default behavior. The designer specializes these classes to deal with behavior specific to the interface, using the inheritance mechanism built into object-oriented languages. These systems can handle highly interactive, direct-manipulation interfaces, because there is a computational links between the input and the output that application can modify to provide semantic processing.

Although these systems make it much easier to create interfaces, they are programming environments and as such are inaccessible to non-programmers. [Ref. 14]

3. Interface-Driven Languages

One of the interface-driven language is Guide. The interface-driven languages are interactive graphical systems for designing and generating graphical user interfaces. They

provide flexibility to the system designer while minimizing the amount of code the designer must write. Their primary goal is to provide a simple, interactive way in which a dialogue developer can specify application interfaces. Once the style of the interface should determined by the developer, he or she should be able to describe with these languages any interface that could be coded by hand.

They provide a great deal of freedom in representing the control path and parameters for action routines. The developer may refer to application constants, types, variables, and function in defining the interface. This ability greatly reduces the number of steps needed to define the interface. Actions are provided to perform application functions based on inputs and application values. Multiple control paths may be represented by the dialogue developer based on inputs, application values, and end-user characteristics. Inclusion of a developer-defined end-user profile allows the developer to represent different interfaces within a single system for different end-users. Various interaction styles and devices can be used, including menus, forms, picking, and keyboard. The developer may choose among any that are suited to a task and may allow the end-user to choose among several styles or devices to provide a particular input. [Ref. 10]

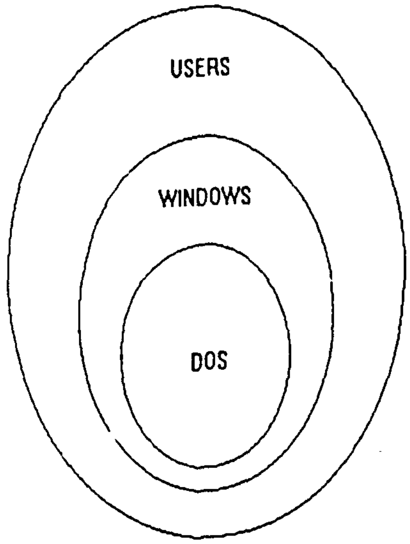

Figure 3.1 Layers for MS-Windows

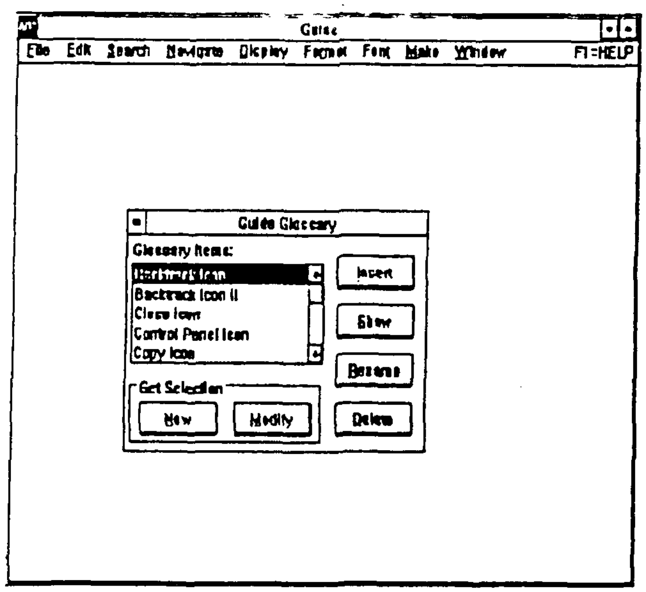

Figure 3.2 Glossary

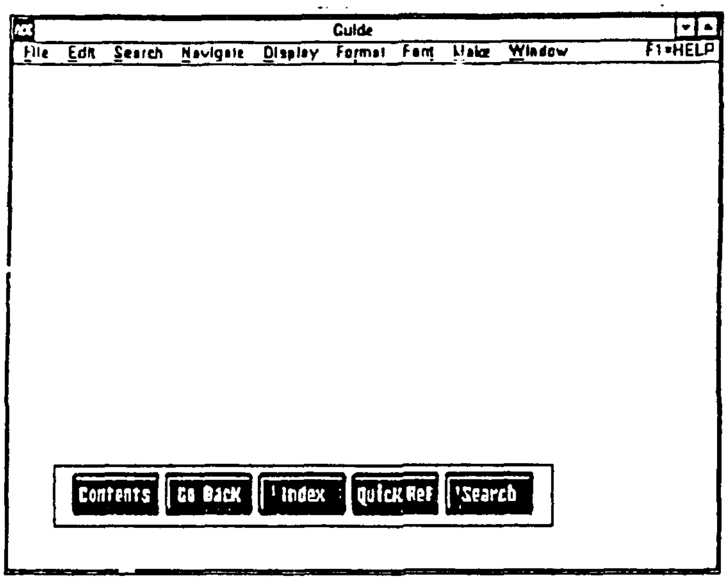

Figure 3.3 Control Panel

IV. DESIGN CONSIDERATIONS

A. GUIDELINES FROM HUMAN FACTORS PERSPECTIVE

The following description is based upon human factors engineering to build efficient and useful user interfaces. The appropriateness of a user interface depends heavily upon the tasks, the users, and the environment. Therefore, no single user interface can be equally appropriate for all applications. But some general guidelines for good user interfaces can be identified with the help of human factors engineering.

1. About Human

During the learning process, users build mental model of how the system behaves. That is, they create representations in their minds of what the system can do, what actions cause what effects, and why those actions cause these effects. Then the models embody the users' understanding of the system. [Ref. 12].

The mental model represents a physical system or software with some plausible cascade of casual associations, reflecting the user's understanding of what the system contains, how it works, and why it works that way. The mental model can be run with trial, exploratory inputs and observed for its resultant behavior.

2. Guidelines

Following guidelines have been suggested to increase the ability of user's mental

models. As such, these guidelines can be helpful in designing user interfaces.

a. *Consistency*

User interfaces must present consistent information. There are two consistencies to consider: internal and external consistencies. Internal consistency is the variance in the behavior of the software itself. The use of modes, where things work differently depending on which mode the user is in, reduces the reception of internal consistency. External consistency is the extent to which the software behavior matches other things the user already knows from other contexts. One way to increase external consistency is providing the user with appropriate metaphors.

b. *Completeness*

Even a simple system must be complete. If the user has to switch often from one program to another just to perform what the user consider to be a single task, the mental model for that system will be more complex.

Providing clear boundaries for the software with a sufficient amount of functionality within those boundaries makes it easier for users to form mental models.

c. *Layering of Functionality*

One way to make a simple appearance of a system is to have layer of functionality. Basic functions are available at the surface layer of the user interface, while putting more advanced functions in deeper layers.

d. *Useful Feedback*

Explanatory error feedback and help messages can help users construct mental models of a system. And, users should be allowed to select the amount of explanation they prefer so that different users can build their mental models at ease.

3. Use of Icons

Human factors engineering strongly recommend the use of icons in user interfaces to represent objects and actions. In general, icons make the interpretation of information on a display more direct and easier to learn and use.

There are some rules or suggestions for building interfaces with icons. [Ref. 13]

a. Make Icons Easy to Use

The meanings of the icons must be learned. What is obvious to the designer may not be obvious to the user. It should be able to develop libraries of tested icons that have been standardized for specific purpose.

b. Avoid Misleading Analogies

If users understand the meaning of an icon, they develop expectations of how they can be used based on their understanding of how things work in the real world. Therefore, we must put the meanings of the real world into icons.

c. Keep Population Stereotypes

A person has his own expectations of how an icon will behavior or can be acted on. Therefore, we have to keep the population stereotypes about compatibility between controls and displays to provide adequate information.

d. *Use for Appropriate Purposes*

In some cases, using icons may not result in a better performance. For example, in calculation, user can do better using numeric key boards than pointing iconized numeric key boards on screen.

B. PRINCIPLES FROM PREVIOUS WORK

The following principles 1 through 6 have been proposed during the development of GLAD. Because the visual interface research is in the early stage, there has not yet been a clear understanding of what constitutes a good user interface [Ref. 3]. And, no single set of interface rules can be optimal for all environments. However, the identification of some design principles will help developers in building better interfaces, and making users accept the interfaces without much confusions, and naturally. Some of these principles are employed in this implementation while some are not.

1. Principle I

"Be able to provide more information when asked."

This principle dictates the inclusion of a simple help system, a confirmation for important processing such as deleting a data file or termination of the running session, explanatory errors, help messages, and so on. When users take an action, the system must provide a feedback to convince them that a desired action has been taken. For example, if users want the screen dump to the printer, the system must provide a dialogue asking how it should be printed, how many copies to be printed, how it should look like

(portrait or landscape), asking go ahead or cancel, and so on. If the system says anything about what it does, then the users will be convinced that their jobs are taken care of. This may be analogous to the saying that the system must provide an appropriate feedback mechanism.

2. Principle II

"Be able to recover from the unintended or erroneous operation."

There must be a mechanism to prevent the system from un intended or erroneous operations so that it maintains consistency. For instance, suppose that a user has removed a data item during the session. And when that user want to exit the system, he must be requested to confirm the removal before exit to operating system.

3. Principle III

"Be able to perform the same operation in more than one ways."

For example, instead of selecting the Quit menu option, users may select the Close option under the system menu box. Or users may double-click the system menu box to exit the system.

By allowing more than one ways of carrying out the same operation, users will be able to use the one that they feel most comfortable with. This helps in supporting a larger number of users. For example, the novice users may prefer using the Quit menu. But as they become more proficient in interacting with the system, they may prefer to close by double-clicking the system menu box.

4. Principle IV

"Be able to perform the logically equivalent operations in a consistent manner."

As a matter of fact, the consistency in the screen representation and in the navigation through an application should always be expected. Therefore, users will easily remember from one use to the next. For example, the same system menu box and the Exit menu choice will appear in every window. If users know how to use them in one window, then they know how to use them in any other window of the system.

5. Principle V

"Be able to display multiple information at the same time."

Providing the capability of displaying multiple information will allow the users to see information in various degree of details.

6. Principle VI

"Be able to display multiple views of the same information."

Multiple display of the same information provides a confidence to users by allowing them to verify the information with another view of the information. For example, if it provides a numerical data, and also a graphical data, users will understand the information clearly and surely.

C. PRINCIPLES DEVELOPED

The following principles have been developed during the implementation of this thesis. Some of principles may overlay with those presented above; but they are intended for highlight somewhat different design goals.

1. Principle VII

"Users can customize the interface."

Users can customize the system environment for easy use of the system. Otherwise the users have to be accustomed to the system that has been developed by others, violating the concept of easy-to-use philosophy.

2. Principle VIII

"The interface must be understood easily and clearly."

It must be stated that a user interface should always have its full meaning and, therefore, it must be understood clearly. If the interface is too complicated, the users need guidelines when to use which and what it means making things even more complicated. For the good user interface, users can understand the meanings and behaviors of interfaces easily and clearly at one glance,like the traffic signs. That is, the user interface must be natural for users.

3. Principle IX

"Do not include interfaces which are useless or have no functions."

Some functions may be redundant with other functions. Some functions may be too hard to use or too difficult to learn how to use, so people simply avoid them. Some functions are not needed: for example, there are navigation icons which include left arrow to go to the previous screen, right arrow to get the next screen, termination icon, a icon which tells it goes to the first screen and to the last screen, and so on; but there can be

a methodology to go to the next screen not by using the right arrow but by using the word(s), phrase, sentence, or graphic element; then the right icon may be useless or less meaningful to use. Such functions should not be included in the user interface.

4. Principle X

"The interface must attract users' attention."

To be a good user interface, the interface must provide a mechanism which make users pay their attention when they are on buttons or any other areas that have some functions or some sort of behaviors. This includes the change of mouse pointers, having different character styles or colors, and so on.

V. IMPLEMENTATION

In this chapter, we discuss the details of the implementation effort.

A. INTERFACE-DRIVEN SOFTWARE DEVELOPMENT TOOL

Interface software is often large, complicated, and difficult to debug and modify. And interface software is difficult to write because frequently it must control many devices, each of which may be sending a stream of input events asynchronously. An application's interface can account for significant amount of codes. The easy-to-use, direct-manipulation interfaces in many modern systems let the user operate directly on objects that are visible on the screen, performing rapid, reversible, incremental actions. Interfaces are not only difficult to create, but there are no design strategies that guarantee the resulting interface will be easy to learn or easy to use. To address the problems, many tools have been created to make interfaces cheaper and easier to design and implement.

1. Classes of Softwares for Building Graphical User Interface

We can classify softwares into two categories with which we can construct graphical user interfaces. They are toolkits, and user interface management systems [Ref. 2, 14].

a. Toolkits

A toolkit is a library of interaction technique. An interaction technique is a

way of using a physical input device to input a value, along with the feedback that appears on the screen. Some of the interaction techniques include menus, scroll bars, and on-screen buttons operated with the mouse. This provides programming abstractions for constructing user interfaces. The examples of these softwares include the X toolkit. These toolkits include objects that are composed of the data to be edited such as text, bitmaps, and more complicated objects such as spreadsheets. These objects can be embedded in multimedia documents. Programmers can specify constraints between objects. The constraints assure that a graphical object stays within a prescribed area or that two visually connected objects stay connected when one or the other is translated.

The disadvantages of using toolkits are that they provide limited interaction styles and are sometimes expensive to create and difficult to use. A toolkit typically includes many interaction procedures that implement many interaction techniques. It is often not clear how to use those procedures to create desired interfaces.

b. User Interface Management Systems

A user interface management system is an integrated set of tools that help programmers create and manage lots of aspects of interfaces. These are, in general, characterized by the separation of the code which implements the user interface to an application from the code for the application itself and the specification of the user interface at a higher level of abstraction than general-purpose programming languages. They minimize the interaction between the application and the interface to maximize their independence. And they usually emphasize abstracting the syntax and semantics of the

user interface.

Its primary purpose is that the interface developers and even end-users can design and modify the interface quickly without requiring experienced programming skills or knowledge of the application. They use special-purpose languages or other representations mechanisms such as finite-state transition diagrams to describe the appearance of the interface and the kinds of interaction it supports. And the specifications are interpreted at run time.

The example of this class includes the Graphical user interface management system. It lets users define the interface, at least partially, by placing objects on the screen with a mouse. Because the visual representation of the interface is one of its most important aspects, a graphical tool is the most appropriate way to specify that representation. It lets the users place interaction techniques such as menus, buttons, and scroll bars on the screen.

2. EVALUATION OF GUIDE AND LOGIIX

Guide is an interface building software that can be run on the IBM personal computer or compatibles. The IBM PC has not provided a powerful environment for graphical user interface without the use of specialized software such as Microsoft Windows. Therefore, Guide has to work through an additional layer of software [Figure5.1]. This makes the application slow, since the command has to be interpreted, and received at each layer.

Users can manipulate text, data, graphics, or combinations of these on screen, and

move easily among different types of information with varying levels of detail without much knowledge of systems software or experience with application development.

As stated in the previous section, Logiix is a special-purpose programming language embedded in Guide. It allows the interface to be built in a simple and intuitive way. Logiix syntax is similar to the Pascal and C programming languages. Logiix provides an event-driven processing. The event is a user's selection of what he/she wants. This provides a flexible structure for building the user interface. That is, what the developer has to do is only specifying what should happen, not how it has to be happened. A Logiix program is a list of instructions contained in the definition of a Guide command button.

One of the strong point of using Guide and Logiix is dynamic data exchange (DDE). It provides a way to communicate with other MS-Windows applications via MS-Windows. Figure 5.2 illustrates how DDE works under MS-Windows. DDE in Logiix provides the interface to allow Guide to act both a DDE client and a DDE server.

a. Compatibility between Guide And Hypercard

Because this implementation has gotten some works from the Argos developed by using Hypercard on an Apple Macintosh, it will be an interesting work to compare between Guide and Logiix, and Hypercard and Hypertalk [Ref. 6].

Guide has a data structure called a document. In a document there is at least one frame of information. Therefore, there can be as many frames as possible. In Hypercard, it provides a stack which consists of cards. A stack is a Hypercard document

and a collection of cards. A card is a basic unit of information in a stack. A card can contain buttons, texts, and graphics.

A button can be placed on a object in a Guide document. As we mentioned above, the buttons can open another document, another frame, or launch another application. Hypercard has the same buttons doing the same work as Guide does. With Guide buttons, we can manipulate the buttons such as the change of the appearance of buttons using pop-up menus. In Hypercard, how a button looks depends on its button style using dialogue box.

We refer to programs behind command buttons as a definition in Guide. And in Hypercard, it is called a script. It is a sequence of English-like statements that respond to events such as the user's clicking on a button. In script, there is a collection of handlers which is a program responding to an event.

In Guide, we can create a link from one object (button) to another which may be in the same document or in another document. A link can be created in Hypercard from a button to a card or stack. After a link is placed, clicking the button takes the same action as Guide button does. We can lock the Guide document by setting a check mark on the Lock Diagram to prevent its contents from unintended or unauthorized changes, still let buttons work. There is a field in Hypercard to lock or unlock to control changes to a stack and a card especially for texts.

A Logiix program is contained in the definition of a Guide command button. Then it is passed to Logiix. The program is compiled into a intermediate stack-based

machine language. Then it is interpreted and executed by a pseudo-machine. The Logiix program is submitted by clicking on a command button, compiled and executed in a single step. Each Logiix program, function, for each button is recognized by the file called GUIDE.S.

In Hypertalk term, a message is an announcement that an event has occurred. A handler is a respond to an event. Event is such as the clicks on a button. Hypercard determines what object the user has acted on and uses this as the address for the message. Then Hypercard will send the message to one of its objects. When an object receives a message, Hypercard searches the object's script for a handler with the same name. When it matches, Hypercard runs any Hypertalk statements in the handler until it encounters an end statement.

B. PORTING IMAGES FROM MACINTOSH

Argos on Apple Macintosh has lots of picture to build its visual interface. Those pictures have been ported to IBM PC compatible for this research. They were uploaded to VAX 11-785, the computer science department's main computer, then downloaded to PC. They are drawn using Macintosh MacPaint. Therefore, we had to convert them to be used on PC. They were converted using a conversion software called The Graphics Link version 1.50 developed by TerraVideo Inc., into Microsoft Windows Paint format with extension MSP. These MSP graphic files were edited by Microsoft Windows Paint to make them bitmap images. After those works, they have been used to make frames and documents.

C. DEVELOPMENT METHODOLOGY

1. Creation of Documents

The fundamental element in this implementation is a document. A document can contain either text or graphics, or both. Several graphics are linked to the document and they are left as external graphics. It is good for developer to change the content of the graphics not touching the document. For example, info.gui has an internal graphic which is a part of its document, and if we want to change the content of the internal graphic, we must create or modify separated graphic file then relink it to the document. However, with external graphic, it is not needed to relink the graphic file to its document. The document uses already established link.

Once we create a document, there must be a definition associated with it. That is, each document has its own definition containing all the definitions in a document [Figure 5.3]. The definition contains programs, in this implementation, a program in Logiix, which determine buttons' behaviors. For the command button, it is a program (script). For example, the definition on the DECK.GUI shows all of its definitions which specifying which button does what [see Appendix A.2].

2. Creation of Buttons

As mentioned in Chapter III, anything that can be selected with the mouse can be made into an button. That is, we can break information into objects and we can attach attributes to them. Attributes determine objects behavior, presentation and how they relate to other objects. If an object is activated by mouse click, we refer to it a button.

To create buttons, we must deactivate all objects using freeze provided in menu. We can place a button on a background graphic in a specific location, not the whole graphic into a button. We can use a transparent graphic element called an invisible which has handles. We can overlay the invisible on a graphic element to make part of graphic element into an object. For example, the frame of BATTLE_GROUP_ZULU is one complete graphic. Each ship model is a separate button created from an invisible overlaid on them [Figure 5.4]. To make an object button, first we select an object then choose a command for type of button we want to make - reference, expansion. note, or command button. For example, on the frame of BATTLE_GROUP_ZULU, all the ships and icons have command buttons. For the object that is surrounded by box, we can navigate through it because the author has defined more detailed frame beyond it. And the rest of objects has not yet been defined for more navigation, therefore they simply show an dialogue box saying "Not Modeled" [Figure 5.5].

3. Programming

Logiix is not an object-oriented language though its environment, Guide, has the concept of an object. Therefore, there is no code sharing between objects. Although two or more buttons take the same action, the same code has to be written for each object. As an example, to print out the window screen dump, we have to write the same program wherever it is needed. However, LOGiiX provides editing feature which is not a code sharing, to do this in different concept. They are copy and paste functions. Once a program is written and the same program is needed elsewhere, we can copy and paste it

to the desired place. It helps developers to reduce their effort in writing the same code repeatedly.

The most common usage of Logiix in this implementation is as the following:

```
#Logiix
function main()
begin
   statement_1;
   ......
   if (comparison) then
   begin
      statement_i;
      ......
   end;
   statement_n;
   ......
end
```

We can use the menu commands provided in Guide such as open, close, print, go to next or previous frame, and so on. The commands operate exactly as if a user selects the menu item manually by clicking on them. It helps the designer to develop easy interface so that users do not have to select from the menu item. The following piece of Logiix program illustrates the usage of these.

```
#logiix
function main()
begin
   return_value = answer(1, "PRINT",
         "Do you want to print it ?");
   if (return_value = 1) then
   begin
      DoMenuId(1008);
   end;
end
```

For the traditional or object-oriented languages, the developer has to program what they want to have for attributes of screen displays, such as cursor pattern, object behavior, title bars, scroll bars, and so on. For example, if we want to have the maximize box on a window, the Actor, an object-oriented language, has to have a following definition:

```
Def create(self, par, wName, rect, style)
{
   ^create(self:Window, par, wName, rect,
    style bitOr WS_MAXIMIZEBOX)
}
```

In Guide, such function is provided within its pop-up or pull-down menus. For example, if we want to display the scroll bar, we use the pull-down menu and set the attribute by simply clicking on it [Figure 5.6].

To implement a dialogue box on the screen, the C++ code should be the following [Ref. 14]:

```
const int space = round(.25*inches);
ButtonState* status;
Frame* frame = new Frame(
   new VBox(
      new VGlue(space, vfil),
      new HBox(
         new HGlue(space, 0),
         new Message("hello world"),
         new HGlue(0, hfil)
      ),
      new VGlue(2*space, 2*vfil),
      new HBox(
         new HGlue(0, hfil),
         new PushButton("goodbye world", status, false),
      new HGlue(space, 0)
```

```
            ),
            new VGlue(space, vfil)
        )
    );
```

Before we write this C++ program, we must have the corresponding object structure as shown in Figure 5.7 [Ref. 14].

But using the Logiix syntax, its equivalent code might be:

```
#logiix
function main()
begin
  messagebox("hello world");
end;
```

4. Creation of Frames

We can think of frames as chapters of a book. That is, frames provide a way to structure a document by dividing it into chunks of related information that is easily manageable. It is really manageable because we can treat a frame as one piece of information, otherwise we must use scrolling bars through an entire document. For example, the document START.GUI consists of several frames which are actually bit-map graphic files such as OPEN.BMP, BATTLE_GROUP_ZULU.BMP,etc [Figure 5.8]. We can handle these graphic files one at a time. However, if we do not make these files frames, and put them all into a document, then we must use scroll bars which is a time consuming and a boring work. Figure 5.9 shows the frames and documents in this implementation. Users can move from one frame to another in sequential order as they are constructed. It is provided in the menu commands such as Previous Frame, Next

Frame, First Frame, and Last Frame. In this implementation, we utilized only the first three commands. Each frame is inserted using insert frame and place menu command. If the inserted file is too big (depending upon the system implemented) to fit into memory, then the system creates a link from a document to that file. Then every time when the file has to be displayed. it is placed into memory, and removed from memory after then.

5. Behaviors of Buttons

In this section, I will explain the behaviors of each button and how they support the principles that satisfy a good user visual interface.

a. The Mouse Pointer Patterns

The mouse pointer pattern changes depending upon where it is placed on the screen. This improves the appearance and reinforce the purpose of an object, and it also bring attention to that object. Figure 5.10 illustrates the change of mouse pointer pattern over a button. And the Figure 5.11 shows how we can change the pattern using set cursor dialogue as it has to be over buttons. This supports the user principle VII, customization. Even if an application developer follows every rules and guidelines of a standard user interface, the usability of the system must still be determined empirically. And standards do not guarantee that people will like an application or be able to use it efficiently. Thus, a system has been developed on expectations of populations and , it is not always true that every user will satisfy in every details. It is necessary for users to customize the system so that they can take full advantage of it. It is easy and interesting

work to customize the system in this working environment as mentioned in subsection 4 above. This supports the principle VIII, the interface must be easily understood, and the principle X, the interface must attract user's attention.

b. Go_Forward

Go_Forward button is indicated by boxed graphic elements or different character styles or both [see Figure 5.3]. Clicking on this button, the system shows more detailed interface about the object (real-world model). This button does exactly what the menu item Next Frame does selected manually. It moves from current frame to next frame. The button on the frame is created by inserting invisible graphic element and overlaying it on the desired position. Its definition looks like:

```
#logiix
function main()
begin
  DoMenuId(1034);
end
```

c. Go_Backward

The icon indicated by a left arrow is used to go backward from the current frame [see Figure 5.3]. It has a command in its document definition like:

```
#logiix
function main()
begin
  DoMenuId(1033);
end
```

With the Go_Forward and Go_Backward, the information or interface is navigated sequentially, except termination in some aspect, as it has been constructed.

This implies the concept of visual interface for database, saying that users navigate through the information database by following the links from one piece of information to the next.

Though this implementation is not complete with database management system, users can move from one piece of information (frame) to another seeking what they wanted.

d. Exit

Users can terminate the running of system and exit to the operating system (MS-DOS). The icon is indicated a left arrow with a vertical bar [Figure 5.12]. Associating with Go_Previous icon, it goes to the first frame then asks if the user really wants to exit to the operating system (MS-DOS). If the user click on the OK button, the system saves all the changes and exits to operating system, otherwise, clicking on the Cancel it remains the running state. For the Logiix program for this button in definition, it looks like:

```
#logiix
function main()
begin
  return_value = answer(1, "EXIT",
    "Do you want to exit the system ?");
  if (return_value = 1) then
  begin
    closeAll(1 + 256);
  end;
end
```

The behavior of this button supports the principle I, the interface must

provide more information when asked, and II, recovery from unintended operation. It asks if a user wants to exit or not, not simply exiting to operating system, Therefore, it provides a mechanism that prevents the user from unintended exit by requesting the confirmation of the user's will.

For the experienced users, they can use the system box or a menu command to exit the system [Figure 5.13]. It is a function provided by an MS-Windows by double clicking on the system menu box or selecting the Close command form File menu. The provides a variety way of operating the system for different level of users. Therefore, it supports the Principle III, performing the same operation in more than one way.

e. Help

Help has been provided for every frame. Figure 5.14 shows the HELP icon and its contents. It just explains the behaviors of each button. This help button opens a document which consists of three frames: help on help, help on document(I), and help on document(II). These three frames are linked forward and backward. And the meanings of the icons on help document are the same as the ones on the regular document described above. This meets the principle IX, the interface must be understood easily. Therefore even the novice users can use the system without much confusion, because they can guess what it means and do what they guess. And it also supports the principle IV, consistent interface. Because the system uses the same interface for the same operation in a consistent manner, the users do not have misunderstanding or confusion when navigating through the system.

f. Print

The Print icon is also shown on every frame. Clicking on this button, it asks if users want to print the current frame out to the printer [Figure 5.15]. If the users answer OK, it shows a print dialogue box. If the users do not want to print out the screen, they can click on the Cancel button. It also supports the Principle I, providing more information when asked, II, recovery from unintended operation, and IV, consistent interface for equivalent operation.

g. Information

In this thesis, the author has implemented the database accessing interface which does not really access a real (relational) database system. It only provides an environment for the future utilization. Therefore, the data that you can get is not from a real database system, but they are actually graphics files created by using Microsoft Paintbrush. And if there must be some changes for these data, then the user must use the Microsoft Paintbrush or compatible graphic softwares.

Clicking on the information icon on the INLET_GEARBOX_ASSEMBLY_BREAKDOWN, denoted by INFO, it invokes another document called INFO_INLET.GUI [Figure 5.16]. The other frame except this one, they do not provide any information. Clinking on those says "Not Modeled" [see Figure 5.4], because it is just an interface which does not access any information database.

D. SAMPLE SESSION

We can run the system by typing "win guide start.gui" at the DOS prompt. Then the

MS-Windows will open the Guide's work environment and again opens the document START.GUI. It displays its first frame in start.gui [Figure 5.17]. Throughout the demo we will see some buttons already made, and we can click on the boxed graphic element to go ahead. Now, lets click on the eye-shaped place. Then it will open the next frame named BATTLE_GROUP_ZULU will be displayed with almost the same interface [Figure 5.18]. Among different ships, there is a ship surrounded by box. Again it has button to go further. Click other ships will cause the system to print a message, Not Modeled [see Figure 5.4]. Clicking on the boxed ship will open the next frame called FFG_7 SIDE PROFILE [Figure 5.19]. On this frame we will see a small ship icon in inversed mode. Clicking on this will open a frame called DECK PROFILE [Figure 5.20]. It is a view of the ship from the sky. To go back to the previous side view frame, we use a return button, left arrow with its tail up. We are using different button to go back at this moment. The top view frame is in different document called DECK_PROFILE.GUI. When we click that inversed small ship, it actually opens another document, not just displaying its next frame in the same document. We can navigate in this fashion from the first frame to the desired frame. Figures 21 through 26 illustrate the navigation path. On this frame, clicking on the info button will open another document called INFOINLE.GUI. Only this button shows some information in this implementation [Figure 27 through 31]. The information has been made using MS-Windows PaintBrush. Because it does not access a real database, there is no way to show what this interface works but this way.

When we want to go back to the previous document, we can use the leftarrow button. It does backtrack from the current frame to the previous frame in the same document which is different from that of return button. This button is seen in almost every frame.

Clicking on the help icon, it opens a document called HELP.GUI [Figure 5.14 and Figure 5.32 through 5.33]. When you finish reading the help messages, you will be returned to the previous document by clicking on the RETURN icon again [see Figure 5.16].

Now lets click on the PRINT icon. It prints the current window out to the default printer as we went over in the above section [see Figure 5.15]. If you want to terminate the session while you are in the system, you can do this by clicking on a button with an arrow and a vertical bar. It will ask you whether you really want to exit to the operating system or not. Suppose we are at DECKPROFILE frame. Then we have two documents opened, one is START.GUI and DECK.GUI which are separate documents. If you click on the EXIT icon to finish your work at this moment, the system takes care of all document that you have been working on. If there has been any changes into any document, the system saves any changes if we click on the yes on the dialogue [Figure 5.34].

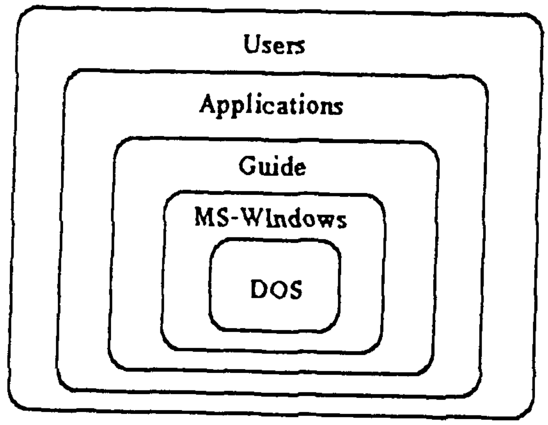

Figure 5.1 Layers for MS-Windows Applications

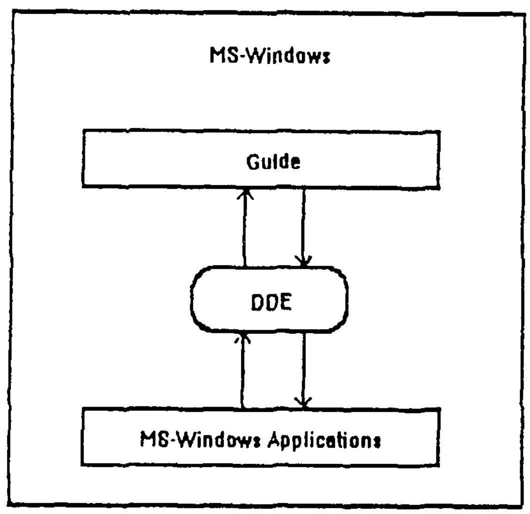

Figure 5.2 DDE under MS-Windows

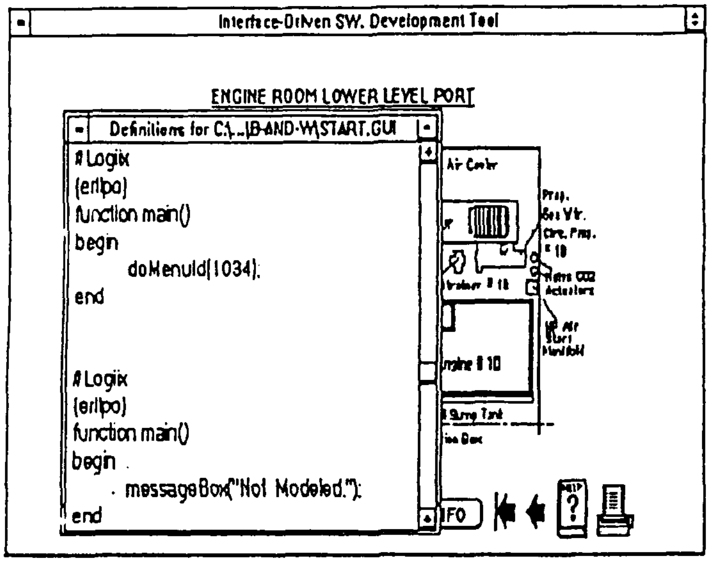

Figure 5.3 Definition in a Document

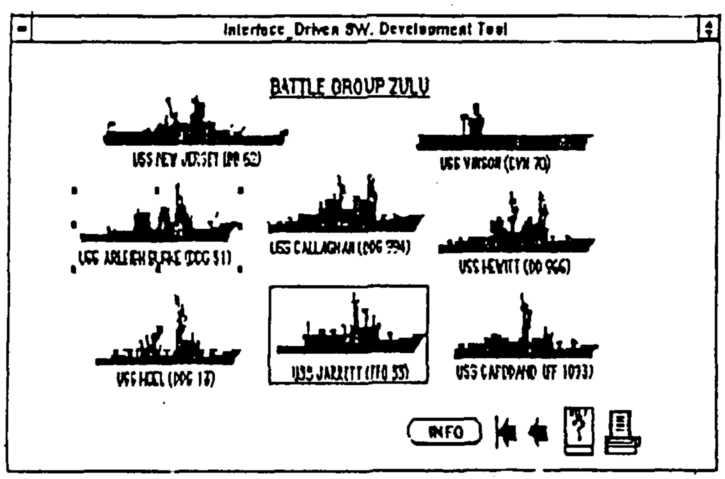

Figure 5.4 Invisible Graphic Element

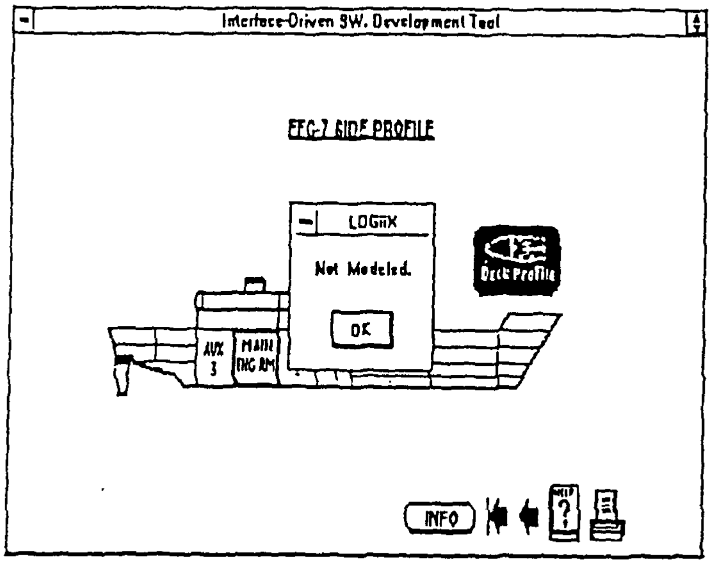

Figure 5.5 Not Modeled Dialogue

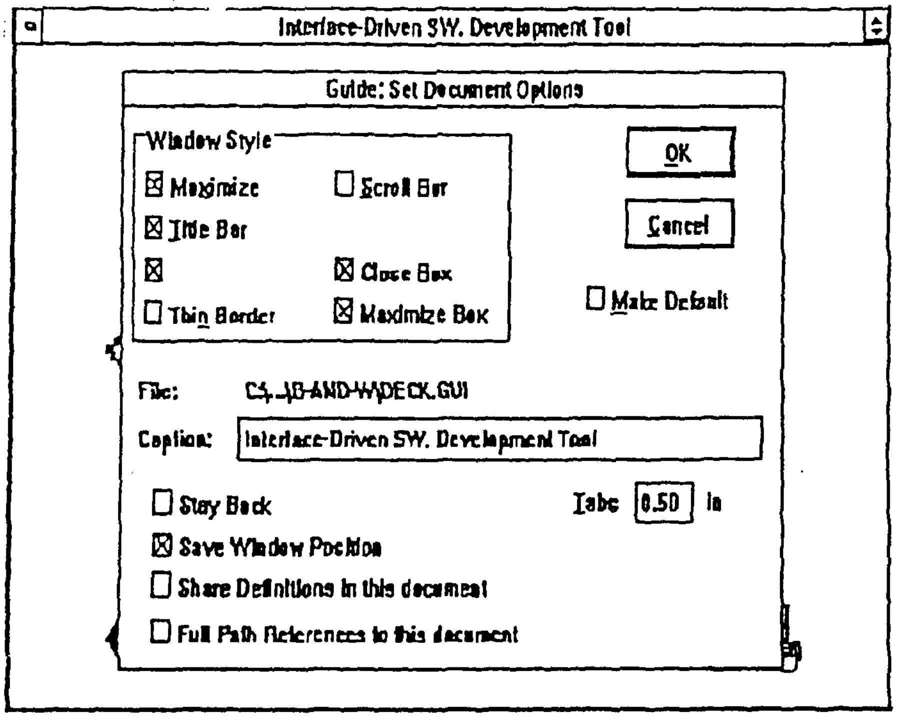

Figure 5.6 Set Document Dialogue

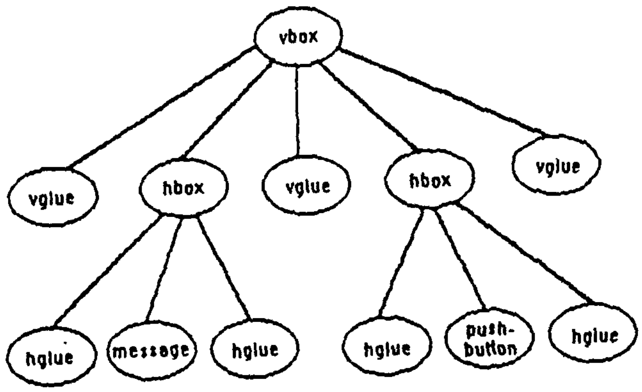

Figure 5.7 Object Structure

```
APL_1.BMP       75026 10/23/90 20:49 .A..
BATT.BMP        71110 10/29/90 21:13 .A..
COSAL_1.BMP     74782 10/23/90 20:49 .A..
DECK.BMP        71110 10/19/90 13:43 .A..
ENGINE.BMP      71110 10/19/90 14:12 .A..
EQUIP_1.BMP     76006 10/23/90 20:50 .A..
ERLLPO.BMP      71110 10/19/90 14:14 .A..
FFG7.BMP        71110 10/19/90 14:15 .A..
FORMS_1.BMP     90934 10/23/90 20:51 .A..
GAS.BMP         71110 10/19/90 14:25 .A..
GTRB.BMP        71110 10/23/90 21:03 .A..
INLET.BMP       71110 10/21/90 23:02 .A..
LM2500.BMP      71110 11/05/90  0:02 .A..
MY_FRAME.BMP    72478 10/29/90 20:55 .A..
OPEN.BMP        71110 10/22/90 13:32 .A..
```

Figure 5.8 Graphic Files Made into Frames

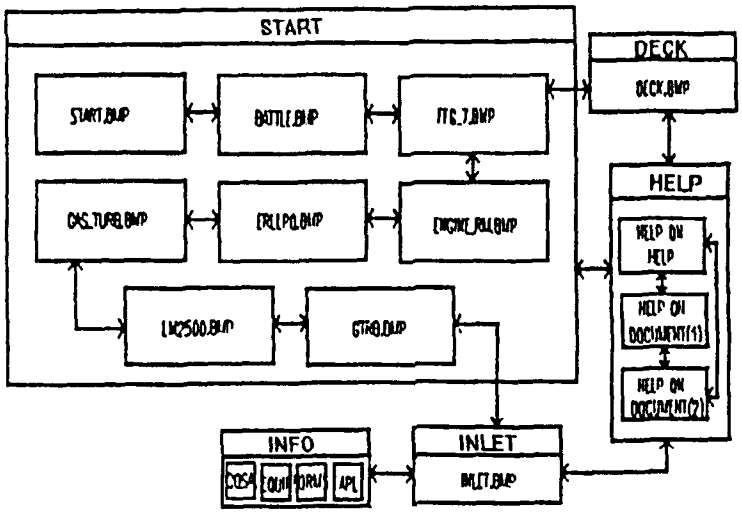

Figure 5.9 Documents and Frames

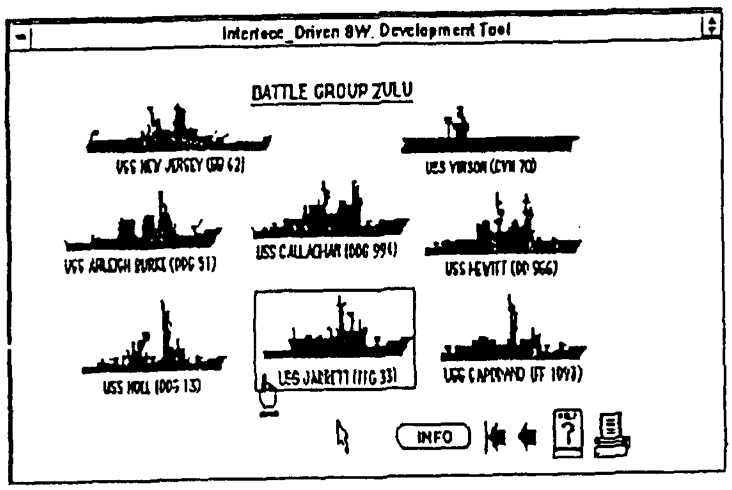

Figure 5.10 Mouse Pointer Pattern Change

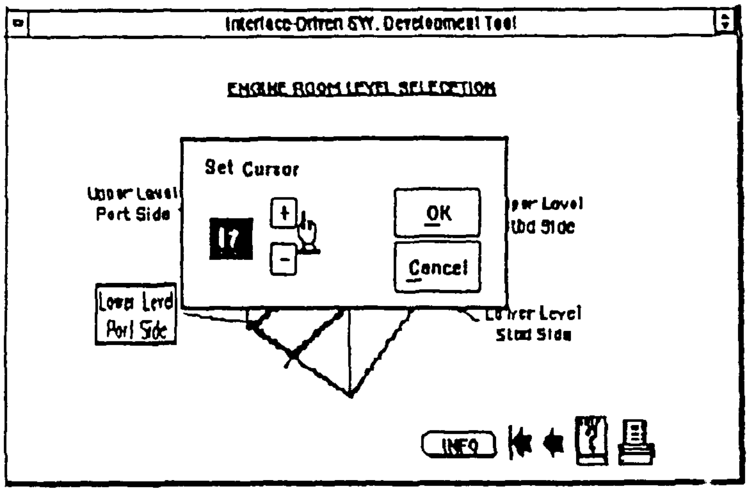

Figure 5.11 Set Mouse Pointer Pattern Dialogue

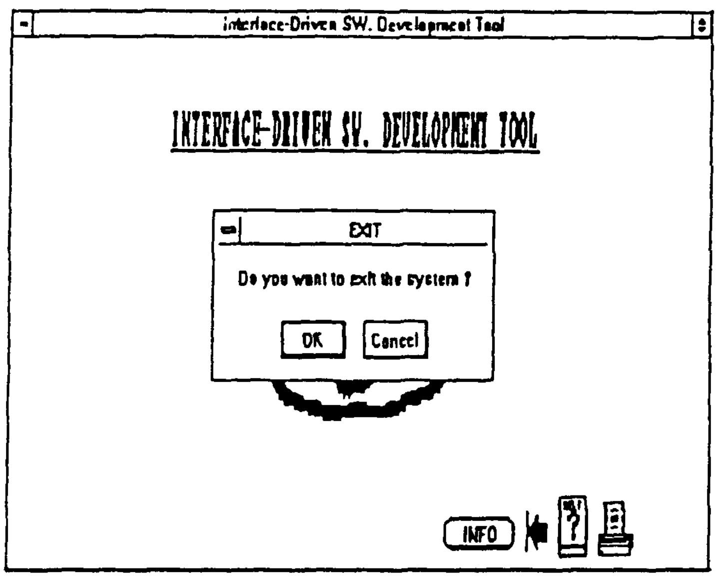

Figure 5.12 Exit Dialogue

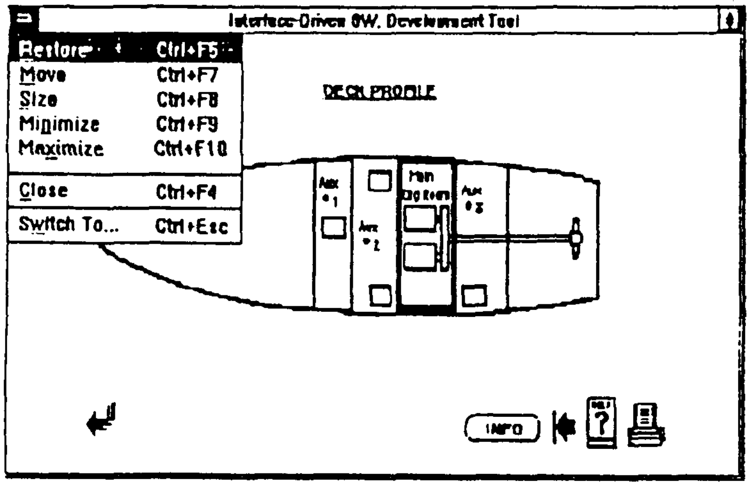

Figure 5.13 System Box

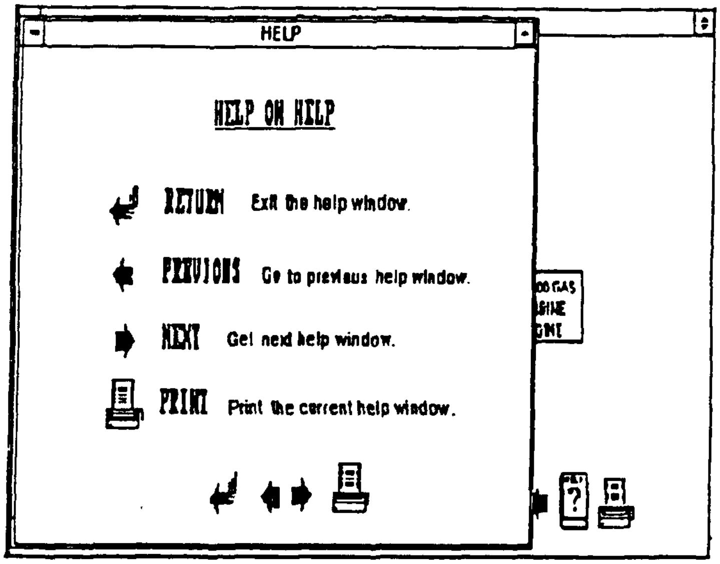

Figure 5.14 Help

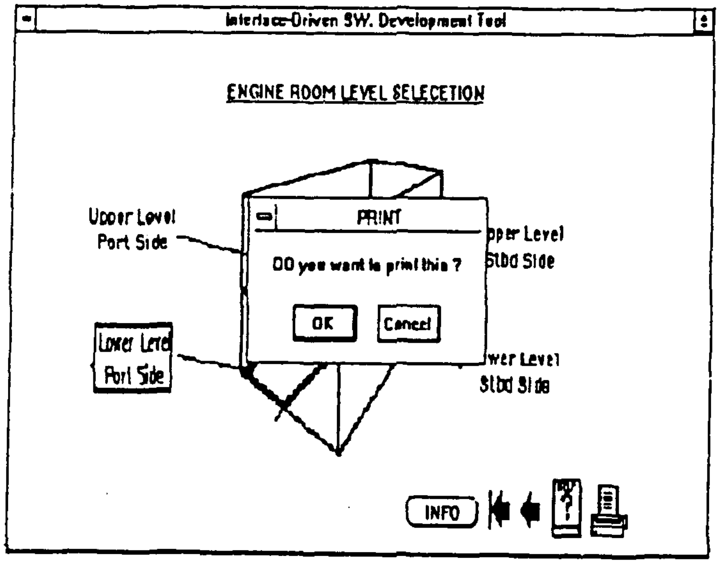

Figure 5.15 Print Dialogue

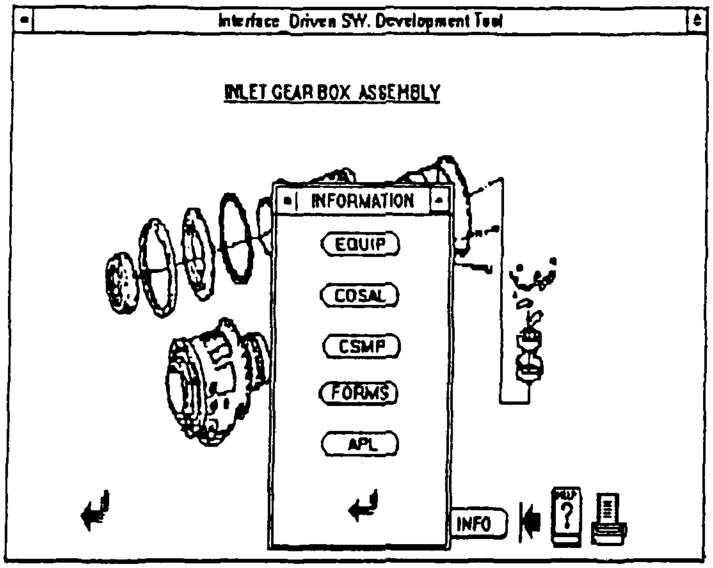

Figure 5.16 Information Document on Inlet Gearbox Assembly Breakdown

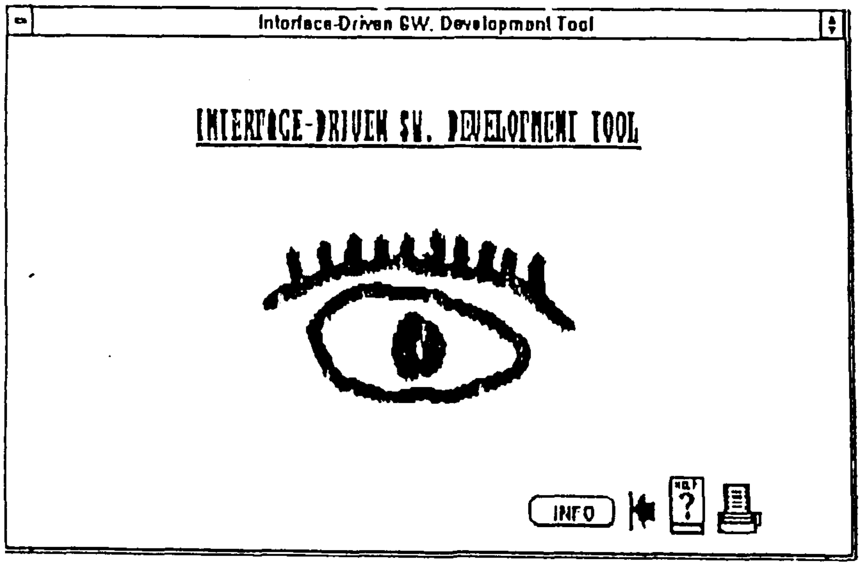

Figure 5.17 The First Frame of Sample Session

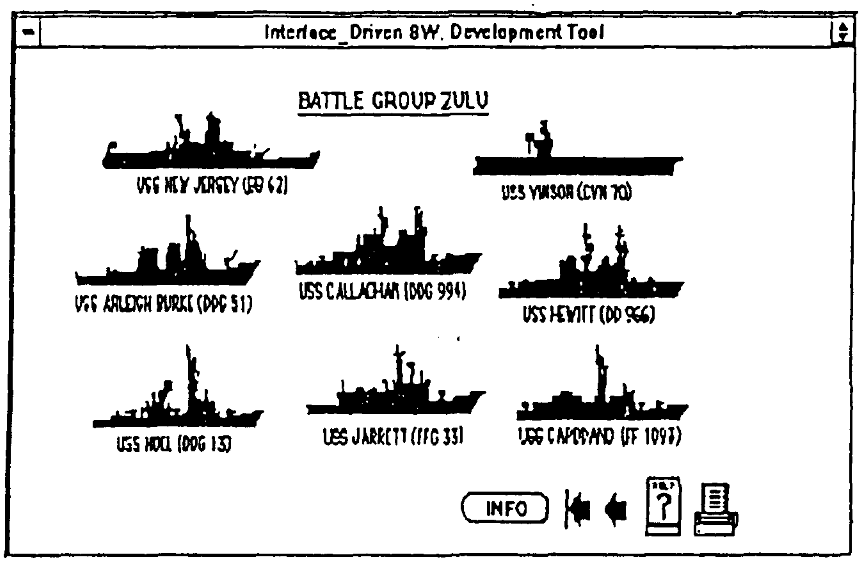

Figure 5.18 Battle Group Zulu

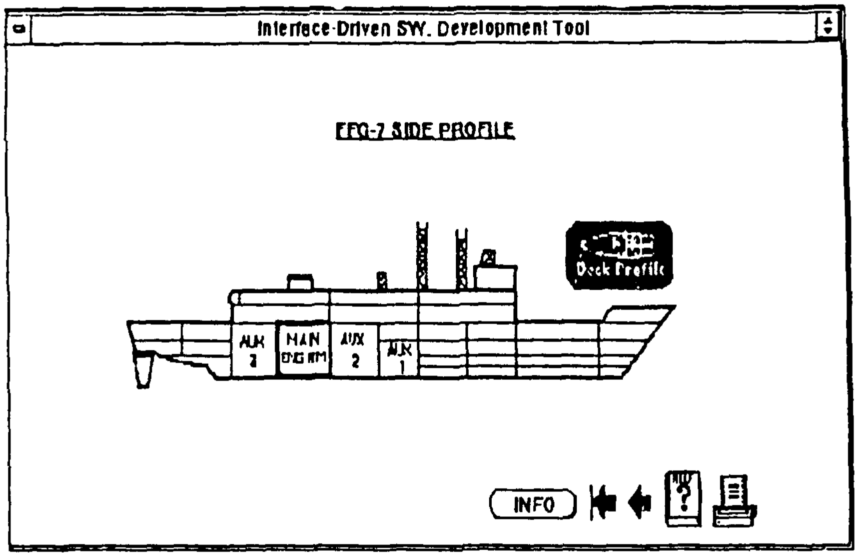

Figure 5.19 FFG-7 Side Profile

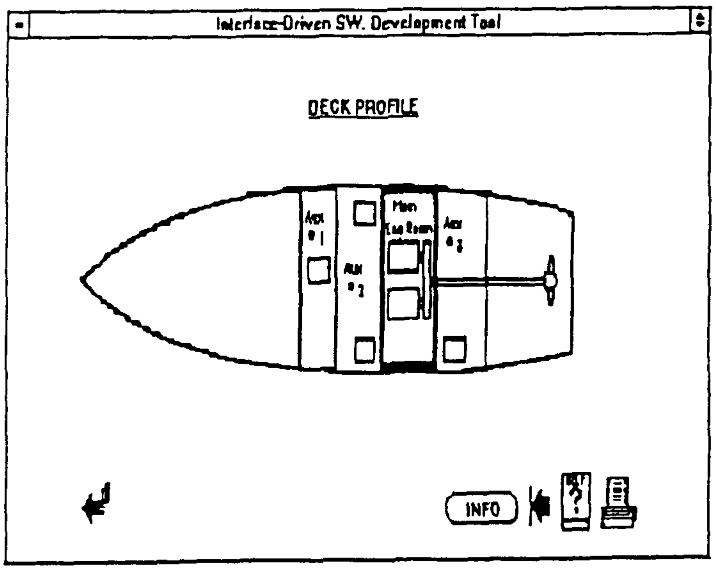

Figure 5.20 Deck Profile

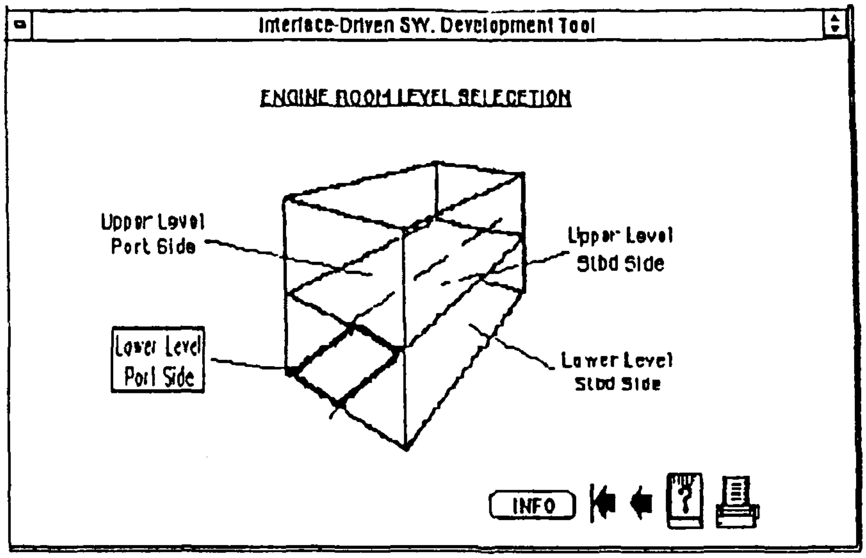

Figure 5.21 Engine Room Level Selection

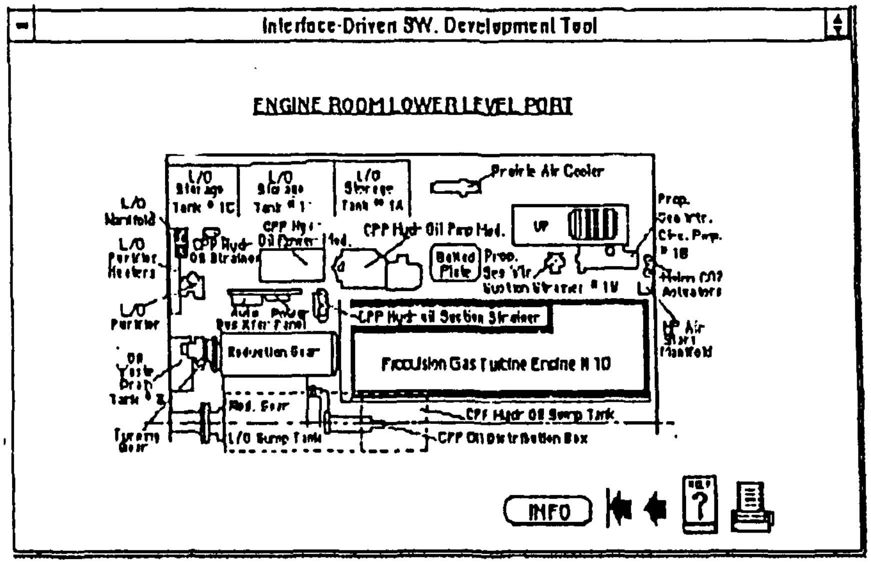

Figure 5.22 Engine Room Lower Port

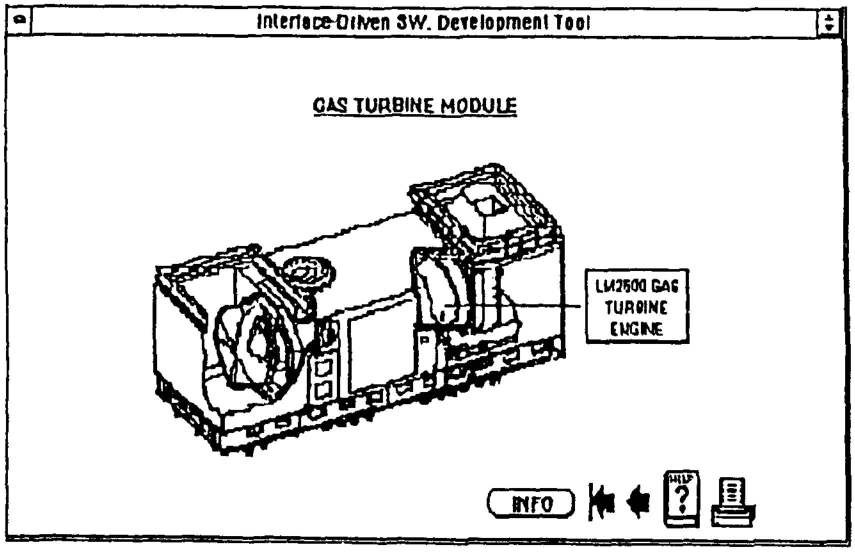

Figure 5.23 Gas Turbine Module

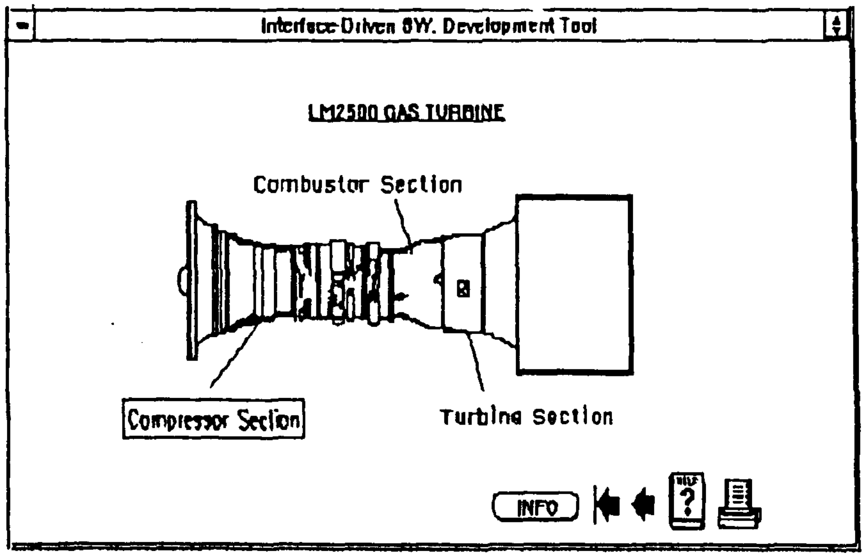

Figure 5.24 LM2500 Gas Turbine

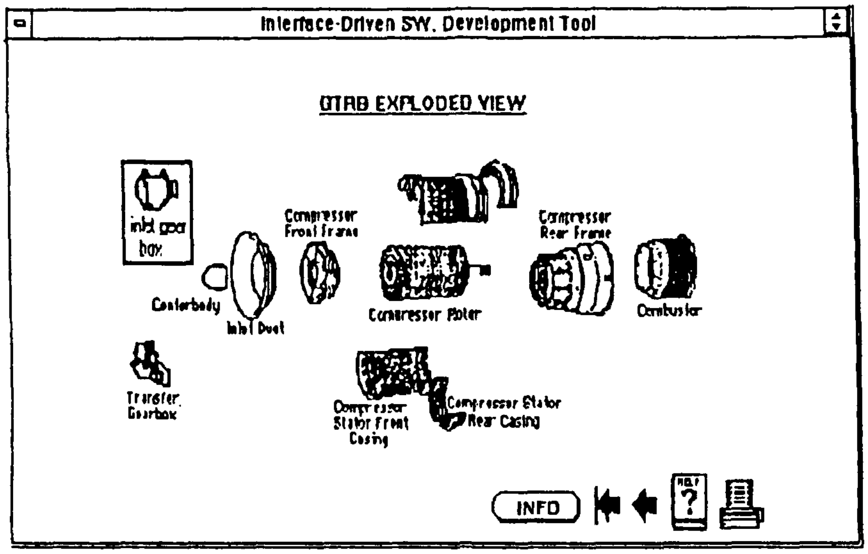

Figure 5.25 GTRB Exploded View

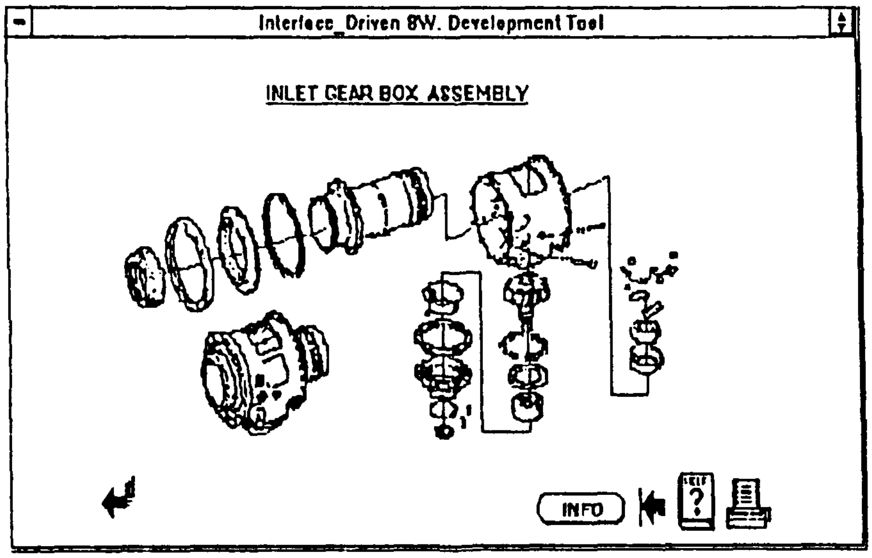

Figure 5.26 Inlet Gearbox Assembly

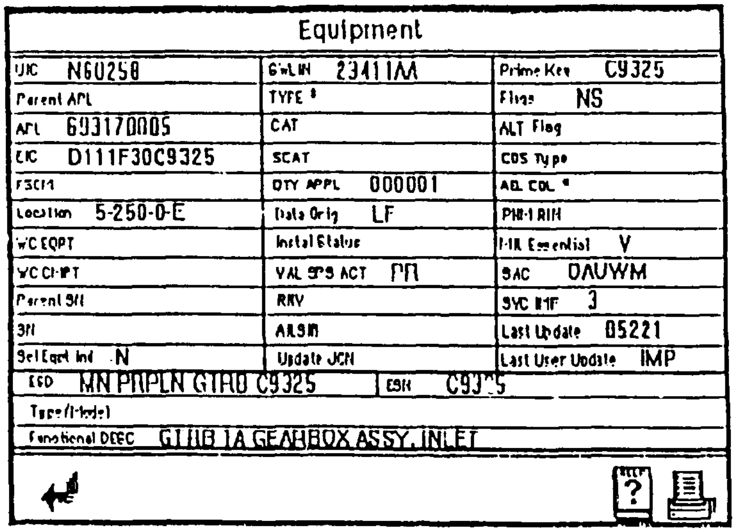

Figure 5.27 Equipment Information

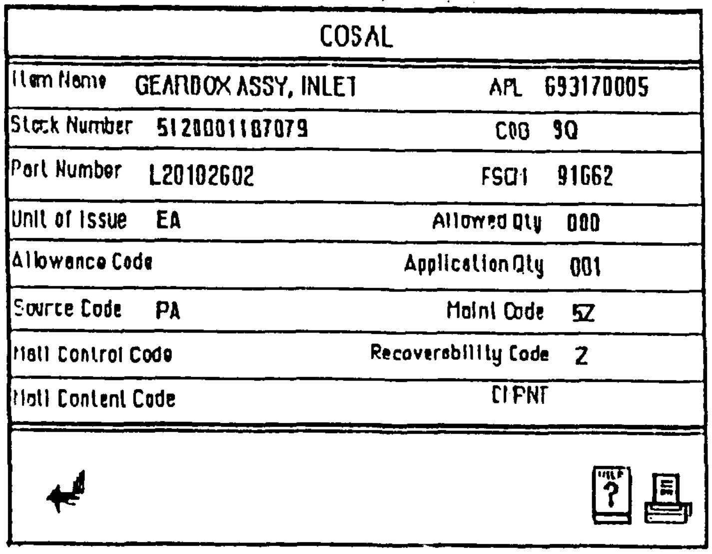

Figure 5.28 COSAL Information

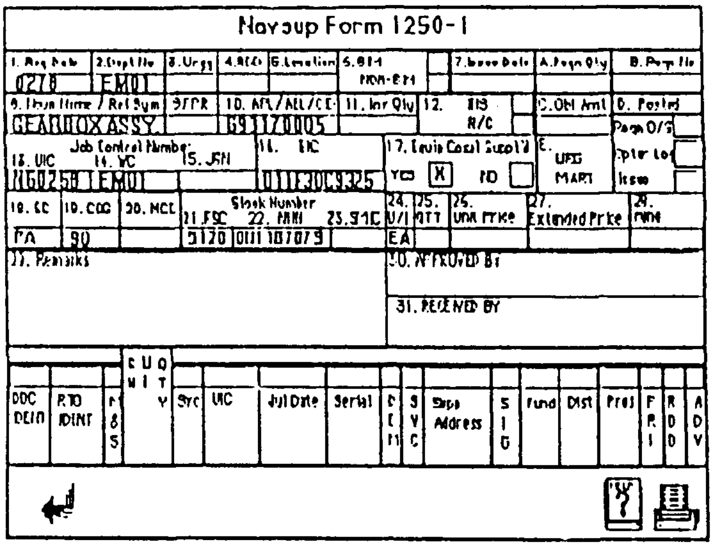

Figure 5.29 Navsup Form 1250-1 Information

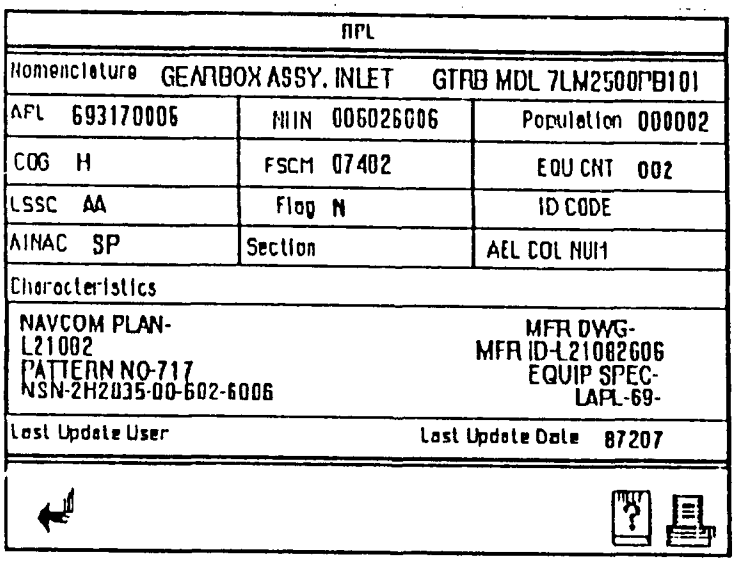

Figure 5.30 APL Information

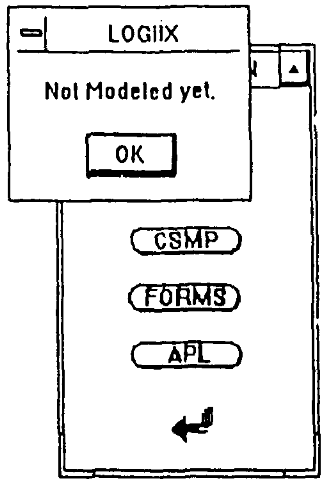

Figure 5.31 Not Modeled dialogue

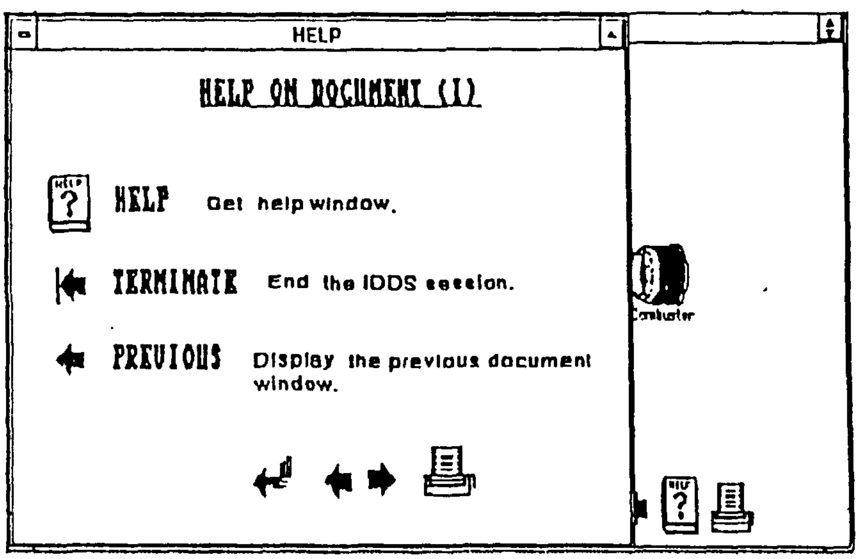

Figure 5.32 Help on Document (I)

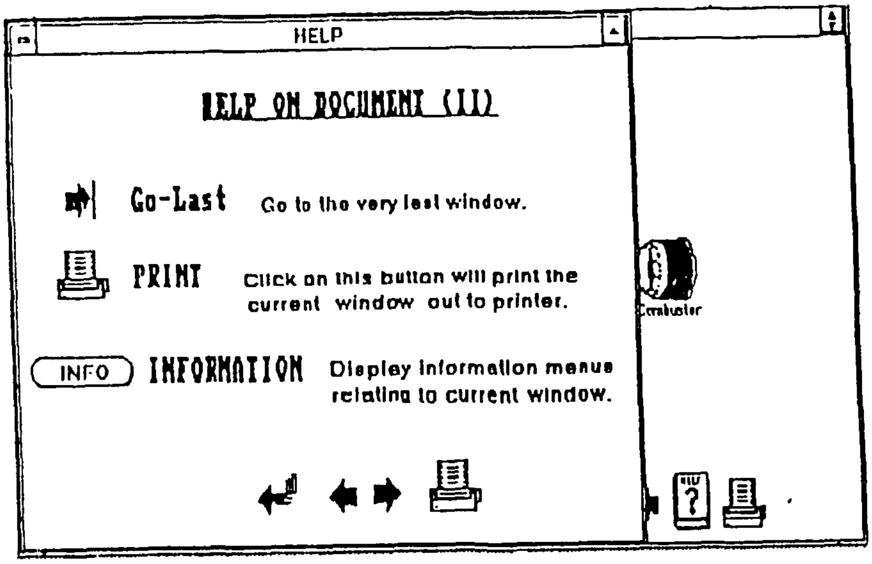

Figure 5.33 Help on Document (II)

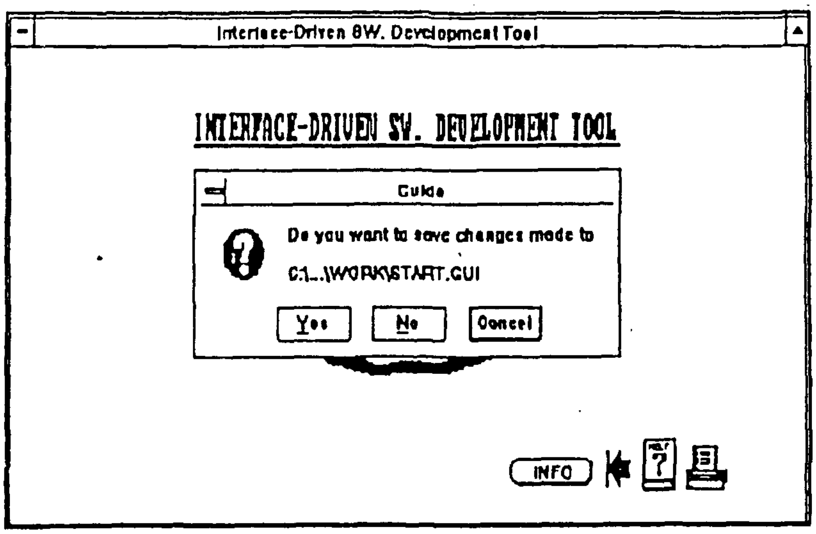

Figure 5.34 Save or Not Dialogue

VI. CONCLUSION

We need tools to keep wp the pace with the increasing amount of information. The traditional database systems are limited in their ability in dealing with a record-based data, and the users have to be familiar with database technology to take a full advantage of database systems as information managers.

Therefore, in this regard, we have built a visual interface with which end-users can manipulate the information being visually represented on the computer screen. So, the users can navigate through the database by following the links from one piece of information to the next. This increases the end-users' paticipation, thus can be a more powerful information.

Since we have employed a windowing software, MS-Windows, we can build graphical interfaces on IBM PC's and compatibles which are widely used. And, since we used an interface-driven software which provides the concept of object, it supports rapid prototyping and incremental development.

The possible following research will be combining the interface frame with real database systems supported by MS-Windows, such as Superbase 4.

APPENDIX A.- PROGRAM LISTS

1. Start

```
#Logiix
{open, go to next frame}
function main()
begin
  doMenuId(1034);
end


#Logiix
{open, ask to exit}
function main()
begin
  ret1 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret1 = 1) then
  begin
    closeAll(1+256);
  end;
end


#Logiix
{open, open help windows}
function main()
begin
  open("help.gui",0,1);
end


#Logiix
{open, ask to print}
function main()
begin
  Ret2 := answer(1,"PRINT", "Do you want to print this ?");
  if (ret2 = 1) then
```

```
  begin
    doMenuId(1008);
  end;
end


#Logiix
{open, dialog box for "info"}
function main()
begin
  messageBox( "Not Modeled.\nNo Information.");
end


#Logiix
{battle, open help windows}
function main()
begin
  open("help.gui",0,1);
end


#Logiix
{battle, ask to print}
function main()
begin
  ret3 := answer(1,"PRINT", "Do you want to print this ?");
  if (ret3 = 1) then
  begin
    doMenuId(1008);
  end;
end


#Logiix
{battle, go to next frame}
function main()
begin
  doMenuId(1034);
```

```
end


#Logiix
{battle, go to previous frame}
function main()
begin
  doMenuId(1033);
end


#Logiix
{battle, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{battle, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{battle, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{battle, display message box}
function main()
begin
```

```
  messageBox("Not  Modeled.");
end


#Logiix
{battle, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{battle, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{battle, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{battle, dialog box for info}
function main()
begin
  messageBox("Not Modeled.\nNo Information.");
end


#Logiix
{battle, ask to exit}
function main()
begin
  doMenuId(1032);
```

```
  ret4 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret4 = 1) then
  begin
    closeAll(1+256);
  end;
end
```

```
#Logiix
{ffg-7, go to previous frame}
function main()
begin
  doMenuId(1033);
end
```

```
#Logiix
{ffg-7, ask to exit}
function main()
begin
  doMenuId(1032);
  ret5 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret5 = 1) then
  begin
    closeAll(1+256);
  end;
end
```

```
#Logiix
{ffg-7, ask to print}
function main()
begin
  ret6 := answer(1,"PRINT", "Do you want to print this ?");
  if (ret6 = 1) then
  begin
    doMenuId(1008);
  end;
end
```

```
#Logiix
{ffg-7, open "deck.gui"}
function main()
begin
  open("deck.gui",0,1);
end


#Logiix
{ffg-7, open help windows}
function main()
begin
  open("help.gui",0,1);
end


#Logiix
{ffg-7, dialog box for info}
function main()
begin
  messageBox("Not Modeled.\nNo Information.");
end


#Logiix
{ffg-7, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{ffg-7, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{ffg-7, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{ffg-7, go to next frame}
function main()
begin
  doMenuId(1034);
end


#Logiix
{engine rm level, go to previous frame}
function main()
begin
  doMenuId(1033);
end


#Logiix
{engine rm level, ask to exit}
function main()
begin
  doMenuId(1032);
  ret7 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret7 = 1) then
  begin
    closeAll(1+256);
  end;
end


#Logiix
{engine rm level, ask to print}
function main()
```

```
begin
   ret8 := answer(1,"PRINT", "Do you want to print this ?");
   if (ret8 = 1) then
   begin
      doMenuId(1008);
   end;
end



#Logiix
{engine rm level, open help windows}
function main()
begin
   open("help.gui",0,1);
end



#Logiix
{engine rm level, dialog box for info}
function main()
begin
   messageBox("Not Modeled.\nNo Information.");
end



#Logiix
{engine rm level, go to next frame}
function main()
begin
   doMenuId(1034);
end



#Logiix
{engine rm level, go to next frame}
function main()
begin
```

```
  doMenuId(1034);
end


#Logiix
{engine rm level, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{engine rm level, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{engine rm level, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{erllpo, go to previous frame}
function main()
begin
  doMenuId(1033);
end


#Logiix
{erllpo, ask to exit}
```

```
function main()
begin
  doMenuId(1032);
  ret9 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret9 = 1) then
  begin
    closeAll(1+256);
  end;
end


#Logiix
{erllpo, ask to print}
function main()
begin
  ret10 := answer(1,"PRINT", "Do you want to print this ?");
  if (ret10 = 1) then
  begin
    doMenuId(1008);
  end;
end


#Logiix
{erllpo, open help windows}
function main()
begin
  open("help.gui",0,1);
end


#Logiix
{erllpo, dialog box for info}
function main()
begin
  messageBox("Not Modeled.\nNo Information.");
end


#Logiix
{erllpo, go to next frame}
```

```
function main()
begin
  doMenuId(1034);
end


#Logiix
{erllpo, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{erllpo, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end



#Logiix
{gas turbine, go to next frame}
function main()
begin
  doMenuId(1034);
end


#Logiix
{gas turbine, go to next frame}
function main()
begin
  doMenuId(1034);
end


#Logiix
```

```
{gas turbine, go to next frame}
function main()
begin
  doMenuId(1034);
end


#Logiix
{gas turbine, go to previous frame}
function main()
begin
  doMenuId(1033);
end


#Logiix
{gas turbine, dialog box for info}
function main()
begin
  messageBox("Not Modeled.\nNo Information.");
end


#Logiix
{gas turbine, ask to print}
function main()
begin
  ret11 := answer(1,"PRINT", "Do you want to print this ?");
  if (ret11 = 1) then
  begin
    doMenuId(1008);
  end;
end


#Logiix
{gas turbine, ask to exit}
function main()
begin
  doMenuId(1032);
  ret12 := answer(1, "EXIT", "Do you want to exit the system ?");
```

```
    if (ret12 = 1) then
    begin
      closeAll(1+256);
    end;
end


#Logiix
{gas turbine, open help windows}
function main()
begin
    open("help.gui",0,1);
end




#Logiix
{lm2500, go to next frame}
function main()
begin
    doMenuId(1034);
end


#Logiix
{lm2500, open help windows}
function main()
begin
    open("help.gui",0,1);
end


#Logiix
{lm2500, go to next frame}
function main()
begin
    doMenuId(1034);
end
```

```
#Logiix
{lm2500, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{lm2500, ask to exit}
function main()
begin
  doMenuld(1032);
  ret13 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret13 = 1) then
  begin
    closeAll(1+256);
  end;
end


#Logiix
{lm2500, dialog box for info}
function main()
begin
  messageBox("Not Modeled.\nNo Information.");
end


#Logiix
{lm2500, ask to print}
function main()
begin
  ret14 := answer(1,"PRINT", "Do you want to print this ?");
  if (ret14 = 1) then
  begin
    doMenuld(1008);
  end;
end
```

```
#Logiix
{lm2500, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{lm2500, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{lm2500, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{lm2500, go to previous frame}
function main()
begin
  doMenuId(1033);
end
```

```
#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{gtrb, open "inlet.gui"}
function main()
begin
  open("inlet.gui",0,1);
end
```

```
#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end
```

```
#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not Modeled.");
end


#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{gtrb, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{gtrb, go to previous frame}
function main()
begin
  doMenuId(1033);
end


#Logiix
```

```
{gtrb, ask to exit}
function main()
begin
  doMenuId(1032);
  ret15 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret15 = 1) then
  begin
    closeAll(1+256);
  end;
end


#Logiix
{gtrb, ask to print}
function main()
begin
  ret16 := answer(1,"PRINT", "Do you want to print this ?");
  if (ret16 = 1) then
  begin
    doMenuId(1008);
  end;
end


#Logiix
{gtrb, open help windows}
function main()
begin
  open("help.gui",0,1);
end


#Logiix
{gtrb, dialog box for info}
function main()
begin
  messageBox("Not Modeled.\nNo Information.");
end
```

2. Deck Profile

```
#Logiix
{deck, return to main program}
function main()
begin
  close(0,1+256);
end


#Logiix
{deck, ask to print}
function main()
begin
  ret17 := answer(1, "PRINT", "Do you want to print this ?");
  if (ret17 = 1) then
  begin
    doMenuId(1008);
  end;
end


#Logiix
{deck, open help windows}
function main()
begin
  open("help.gui",0,1);
end


#Logiix
{deck, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end



#Logiix
```

```
{deck, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{deck, display message box}
function main()
begin
  messageBox("Not  Modeled.");
end


#Logiix
{deck, open information window}
function main()
begin
open("info.gui",0,1);
end


#Logiix
{deck, ask to exit}
function main()
begin
  ret18 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret18 = 1) then
  begin
    closeAll(1+256);
  end;
end
```

3. Inlet Gearbox Assembly Breakdown

```
#Logiix
{inlet, return to main program}
function main()
begin
  close(0, 1+256);
end


#Logiix
{inlet, open information window}
function main()
begin
  open("infoinle.gui",0,1);
end


#Logiix
{inlet, ask to exit}
function main()
begin
  ret19 := answer(1, "EXIT", "Do you want to exit the system ?");
  if (ret19 = 1) then
  begin
    closeAll(1+256);
  end;
end


#Logiix
{inlet, ask to print}
function main()
begin
  ret20 := answer(1, "PRINT", "Do you want to print this ?");
  if (ret20 = 1) then
  begin
    doMenuId(1008);
  end;
end
```

```
#Logiix
{inlet, open help window}
function main()
begin
   open("help.gui",0,1);
end
```

4. Help

```
#Logiix
{3, go to next frame}
function main()
begin
  doMenuId(1034);
end


#Logiix
{3, go to previous frame}
function main()
begin
  doMenuId(1033);
end


#Logiix
{2, go to previous frame}
function main()
begin
  doMenuId(1033);
end


#Logiix
{2, go to next frame}
function main()
begin
  doMenuId(1034);
end


#Logiix
{1, go to previous frame}
function main()
begin
  doMenuId(1033);
end
```

```
#Logiix
{1, go to next frame}
function main()
begin
  doMenuId(1034);
end


#Logiix
{1, return to main program}
function main()
begin
  close(0,1+256);
end


#Logiix
{2, return to main program}
function main()
begin
  close(0,1+256);
end


#Logiix
{3,return to main program}
function main()
begin
  close(0,1+256);
end


#Logiix
{1, ask to print}
function main()
begin
  ret21 := answer(1,"PRINT", "DO you want to print this ?");
  if (ret21 = 1) then
  begin
    doMenuId(1008);
```

```
  end;
end


#Logiix
{2, ask to print}
function main()
begin
  ret22 := answer(1,"PRINT", "DO you want to print this ?");
  if (ret22 = 1) then
  begin
    doMenuId(1008);
  end;
end


#Logiix
{3, ask to print}
function main()
begin
  ret23 := answer(1,"PRINT", "DO you want to print this ?");
  if (ret23 = 1) then
  begin
    doMenuId(1008);
  end;
end
```

5. Information on Inlet Gearbox Assembly

```
#Logiix
{info_inlet}
function main()
begin
  open("cosal_1.gui",0,1);
end
```

```
#Logiix
{info_inlet}
function main()
begin
  messageBox("Not Modeled yet.");
end
```

```
#Logiix
{info_inlet}
function main()
begin
  open("forms_1.gui",0,1);
end
```

```
#Logiix
{info_inlet}
function main()
begin
  open("apl_1.gui",0,1);
end
```

```
#Logiix
{info_inlet}
function main()
begin
  close(0,1+256);
end
```

```
#Logiix
{info_inlet}
function main()
begin
   open("equip_1.gui",0,1);
end
```

LIST OF REFERENCES

1. Andrew Monk, "Fundamentals of Human-Computer Interaction," Academic Press, Inc., 1984.

2. Brad A. Myers, "User-Interface Tools: Introduction and Survey," IEEE Software, Jan. 1989.

3. C. Thomas Wu, "Development of a Visual Database Interface - An Object-oriented Approach," NPS. Monterey, CA.

4. Duff, C., et al, "Actor Language Manual," The Whitewater Group, Inc., 1989.

5. David Maier, Zacob Srein, Allen Otis, Alan Purdy, "Development of an Object-oriented DBMS," ACM, 1986.

6. Dan Shafer, "HyperTalk Programming," Hayden Books, 1989.

7. Dave Thomas, "What's in an Object," Byte, March 1989.

8. George Copeland, David Maier, "Making Smalltalk a Database System," ACM, 1984.

9. "Guide 3.0 Reference Manual," Precision. Inc., 1990.

10. H.R. Hartson and D. Hix, "Human-Computer Interface Development," ACM Computing Surveys, Vol.21, No.1, March 1989.

11. J. Banerjee, H. Chou, Jorge F, Garza, W. Kim, D. Woelk, N. Ballou, and H. Kim, "Data Model Issues for Object-Oriented Application," ACM Transactions on Office Information Systems, Vol. 5, No. 1, Jan. 1987.

12. Kathleen Potosnak, "Mental Models: Helping Users Understand Software," IEEE Software, Sept. 1989.

13. Kathleen Potosnak, "Do Icons make User Interface Easier to Use ?" IEEE Software, May 1988.

14. Mark A. Linton, John M. Vlissides, and Paul R. Calder, "Composing User Interfaces with InterViews," IEEE, Feb. 1989.

15. Mahesh H. Dodani, Charles E. Hughes, and J. Michael Moshell, "Separation of Powers," Byte, March 1989.

16. Thomas Atwood, "Applying the Object Paradigm to Databases," Computer Language, Sept. 1990.

17. William Geobel Anthony Sympson III, "Graphics Interface for Attribute Based Data Language Queries from a Multi-lingual, Multi-model, Multi-Backend Database system over an Ethernet Network," Masters Thesis, Naval Postgraduate School, Monterey, Ca., Dec. 1989.

INITIAL DISTRIBUTION LIST

1. Defence Technical Information Center — 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Dudley Knox Library — 2
   Code 52
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Curriculum Office, Code 37 — 1
   Computer Technology
   Naval Postgraduate School
   Monterey, California 93943-5002

4. Professor C. Thomas Wu (Code CS/Wq) — 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5002

5. Professor Myung W. Suh (Code AS/Su) — 1
   Administration Science Department
   Naval Postgraduate School
   Monterey, California 93943-5002

6. Professor Kyung-Chang Kim (Code CS/Ki) — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5002

7. Professor Dong-Soo Kim (Code ME/Km) — 1
   Mechanical Engineering Department
   Naval Postgraduate School
   Monterey, California 93943-5002

8. Captain Jae-Du Jung | 1
SMC 1504, Naval Postgraduate School
Monterey, California 93943-5002

9. Captain Jung-Hyun Park | 1
SMC 1818, Naval Postgraduate School
Monterey, California 93943-5002

10. Korea Military Academy Library | 1
P.O. Box 77, Gong-Neung Dong,
No-Won Gu, Seoul,
South-Korea, 132-240

11. Army Central Library | 1
Army Heaquarter, Bu-Nam Ri,
Du-Ma Myeon, Non-San Gun, Chung-Nam,
South-Korea, 320-919

12 Captain Heung-Taek Kim | 1
267-9, Kan-Seok Dong, Nam-Dong Gu,
In-Cheon, South-Korea